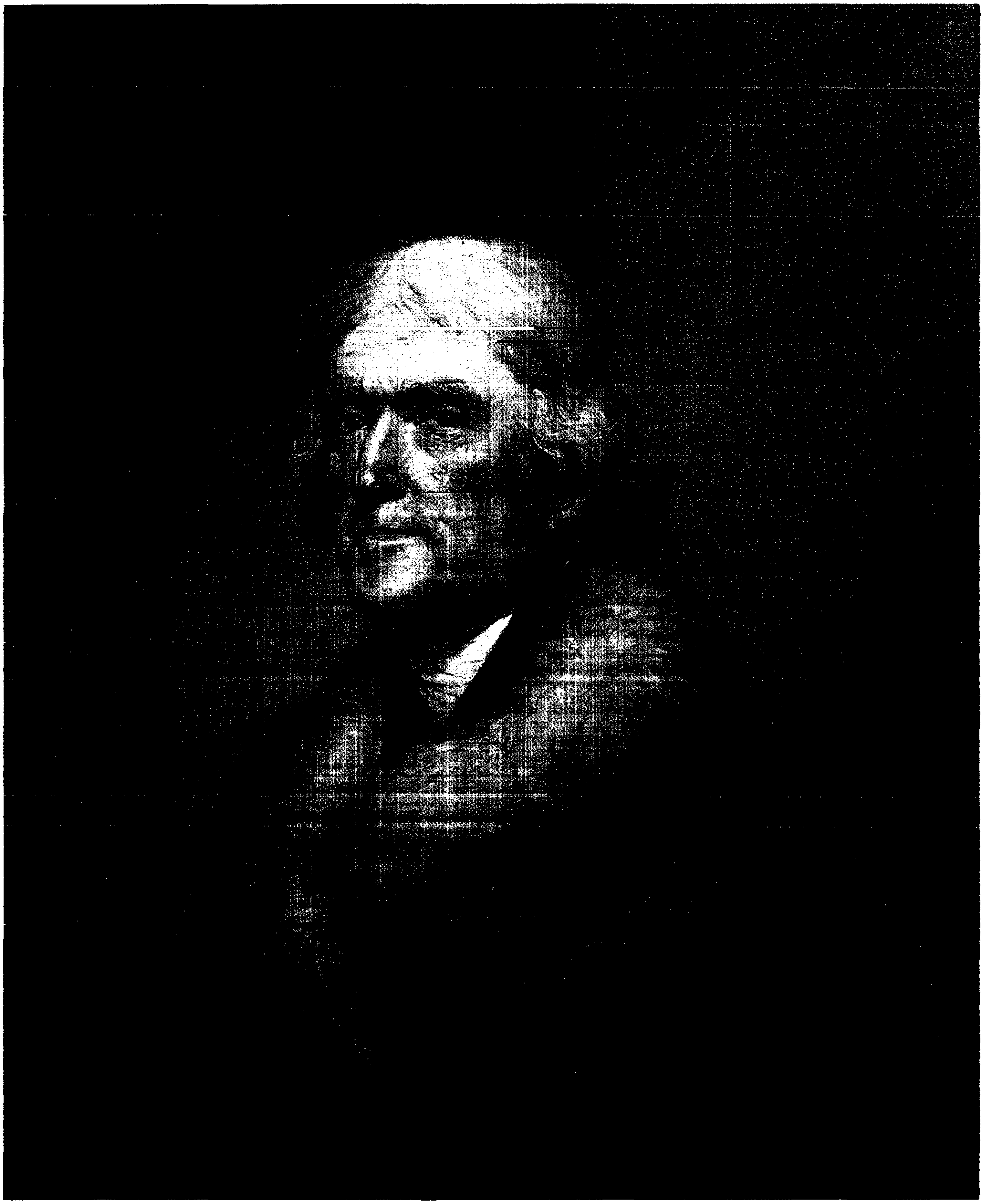


This bicentennial edition of the Proceedings of the 1976 National Computer Conference is dedicated to the memory of

THOMAS JEFFERSON

who wrote,

I have sworn upon the altar of God
eternal hostility against every form
of tyranny over the mind of man.

AFIPS
CONFERENCE PROCEEDINGS

1976
NATIONAL COMPUTER CONFERENCE

Member National Bicentennial
Service Alliance

June 7-10, 1976

New York City, New York

STANLEY WINKLER
Editor and Program
Chairman

CARL HAMMER
Conference Chairman

AFIPS PRESS
210 SUMMIT AVENUE
MONTVALE, NEW JERSEY 07645

The ideas and opinions expressed herein are solely those of the authors and are not necessarily representative of or endorsed by the 1976 National Computer Conference or the American Federation of Information Processing Societies, Inc.

Library of Congress Catalog Card Number 55-44701
AFIPS PRESS
210 Summit Avenue
Montvale, New Jersey 07645

Printed in the United States of America

Preface

by CARL HAMMER
Conference Chairman
Sperry Univac
Washington, DC

This is the second time that I have had the privilege of serving the computing community by assuming a role of major responsibility in a National Computer Conference. Dr. Harvey Garner, General Chairman of the First National Computer Conference, asked me in the fall of 1972 to manage the Science and Technology Program for "his" '73 NCC in New York. Three years later, during the Nation's Bicentennial Year, I have returned to serve as Conference Chairman of "my own" '76 NCC.

This conference was carefully planned as a profound educational experience for all in attendance. We used an innovative approach throughout to make it a Landmark Event long to be remembered. Our programmatic activities include many unusual events such as hands-on demonstrations in computer networking, student computer projects from all over the country, computer graphics art exhibits from all over the world. We obtained some entirely new materials for our science film theater and we tailored special programs for the convenience of the handicapped.

The preparations for this conference commenced in the fall of 1974. We recruited a talented and dedicated team of professional volunteers to help us plan and manage it. The members of this Conference Steering Committee were our brains—and often also our arms and legs. It was *their* unbounded enthusiasm and *their* unflagging spirit which put this magnificent show on the road. More than once many of them toiled around the clock, or gave up weekends and even holidays. The community owes them much for their dedication to the cause. . . .

The transient nature of even the most successful conferences is not likely ever to change. But some of them, including these National Computer Conferences, make a permanent contribution in terms of the archival records of their Proceedings. Here we capture for posterity the most current reports on recent achievements and new applications, on advances at the frontiers of computer science and technology. We are justifiably proud of this volume which contains the papers that were selected for delivery at this conference. We acknowledge with deeply felt gratitude the leadership role of our program chairman, Dr. Stanley Winkler, who structured this exciting program and who assembled these proceedings. We who worked with him on the colossal task of designing this meaningful and balanced program will never forget this experience which has enriched our lives and strengthened many personal bonds as well.

We have striven to give this conference and these proceedings a quality which is appropriate for the occasion. As our country prepares for her bicentennial celebration we are also observing the hundredth birthday of the telephone without which interactive computing and distributed networks would be inconceivable. We remember with nostalgia the First (Inaugural, one might say) Joint Computer Conference held twenty-five years ago in Philadelphia; hopefully our efforts will be judged worthy of its great tradition and of all past FJCCs, SJCCs and NCCs. Another milestone comes to mind for 1951: During that memorable year the first commercially built computer was delivered to the U. S. Bureau of the

Census. 1776, 1876, 1951 were evidently years in which men of great vision pioneered unforgettable events with much impact. If indeed the "Past is Prologue," we should forever be motivated and inspired by this rich legacy of our nation and the computing profession.

For developing the stimulating materials which comprise these proceedings we are deeply indebted to our stalwart program chairman, Dr. Stanley Winkler. We are grateful to the circa two thousand persons who contributed to this effort by writing or reviewing these papers, or by participating in the program sessions as organizers, discussants and speakers. We thank all Conference Steering Committee members for giving so unstintingly of their time and resources; we also thank their employers or sponsors for allowing them to draw so heavily on their resources. We are also grateful for the support received from the AFIPS staff who most graciously coped with our many idiosyncrasies and scheduling difficulties. Finally, it is a pleasure to acknowledge the guidance we received from the NCC Committee and Board. The names of all who took part in this herculean effort are recorded in this monument to their tenacity and endurance. This was truly a team effort and it was well worth it.

As we commemorate the first twenty-five years of electronic data processing we observe that the introduction of computers into our society has already caused profound changes in everyone's life style. Digital communications today provide public access to the power of computers as readily as the earlier telephone facilitated human dialogue. Global communications systems span the earth as we probe the depths of our solar system and even of the universe. Electronic miniaturization is revolutionizing entire industries and radically new concepts of electronic systems architecture are evolving. Computers have become a new source of power, facilitating the transition from traditional management systems to those of a society which is data and information rich. Yet, as these Proceedings establish so well, we are still at the very threshold of electronic invention and innovation!

As we continue along a path of near-exponential progress—and there is little reason to doubt that we will do so for quite some time—the pervasiveness of electronic systems and their impact on societal structures is bound to exceed our cumulative experience with all earlier technological developments by several orders of magnitude. Whatever one cares to read into such prophecies, human values and the attainable quality of life for all mankind will and must emerge as the ultimate beneficiaries. Perhaps this is the greatest reward which posterity can bestow on us as we place this volume into the public domain. Hopefully the concepts and results espoused herein will help liberate mankind from the self-imposed yokes of rote and drudgery, ushering in a brighter future that knows how to make human use of human beings. . . .

CONTENTS

INTRODUCTION

COMPUTERS AND PEOPLE

SOCIETAL CONCERNS

THE COMPUTER PROFESSION

ISSUES IN COMPUTING

APPLICATIONS SERVING PEOPLE

SYSTEMS

COMPUTER SYSTEMS

SYSTEMS MANAGEMENT

NETWORKING

BUSINESS AND INDUSTRY SYSTEMS

SCIENCE AND TECHNOLOGY

COMPUTER AND DATA BASE ARCHITECTURE

SOFTWARE

COMPUTER SCIENCE

APPLICATIONS OF COMPUTER SCIENCE

INTRODUCTION

A view of the world of computing as seen at the 1976 National Computer Conference

by STANLEY WINKLER
IBM Corporation
Gaithersburg, Maryland

ABSTRACT

The twenty-fifth anniversary of joint computer conferencing and the bicentennial of the United States of America are celebrated during this 45th in a series of joint conferences. The Conference also commemorates the 25th anniversary of the introduction of commercial computing. The state of the computer profession and industry is mirrored in the conference program and these conference proceedings are a selected distillation of the program. The spirit of the American Revolution is reflected in the attention given to Computers and People in general and Societal Concerns in particular. About one third of the Conference is devoted to Computers and People and the remaining two thirds is divided almost equally between Systems and Science and Technology. The quality, scope and diversity of the papers in this volume, as they represent the state-of-the-art today, augurs well for the future.

INTRODUCTION

June 7, 1976, the opening day of this Conference, is the 200th anniversary of the introduction, by Richard Henry Lee, of the resolution for independence of the United States of America. It is, thus, proper in a bicentennial year that we recognize this connection with the American Revolution, and the dedication to Jefferson provides that recognition. However, in dedicating this volume to the memory of Thomas Jefferson, I also hoped to invoke that fierce spirit of the man who swore eternal hostility against tyranny over the mind of man. As we look at the state of our profession in the mirror of this conference, it is easy to see a maturity in the realm of technological capability. The equivalent maturity in the understanding of social impacts and public policy direction is not so easy to detect. Perhaps it is appropriate to recall the dictum of Norbert Wiener who wrote, ". . . danger to society is not from the machine but from what man makes of it."[1]

During our conference, we celebrate not only the nation's bicentennial, but also the 25 year anniversary of joint conferencing and the 25th anniversary of the introduction of commercial computing. On December 10–12, 1951, a relatively homogenous group met in Philadelphia to discuss the characteristics and performance of ten working, large-scale electronic digital computers. In many ways it was a remarkable meeting as may be seen from the contents of that first proceedings.[2] The Keynote Address for 1951 by W. H. MacWilliams of the Bell Telephone Laboratories is a very interesting sketch of the past, present and future of the computing industry as seen in 1951. The last two papers in that first program were a discussion of the applicability of transistors to digital computation, by J. H. Felker and a forecast of the future by J. W. Forrester. Elsewhere in this volume, Herb Grosch provides us with an highly personal view of that first conference and the succeeding twenty-five years of joint computer conferencing in which he captures the gestalt of these conferences.[2]

The program for this 45th Conference is structured into three areas. Each in a sense is a conference within the conference and each area is further divided into four affinity groupings or tracks. These are shown in the Conference-at-a-Glance which is reproduced here as Figure 1. The program is intended to mirror the current state of our profession and industry and this volume of conference proceedings is a selected distillation of the program.

PAST IS PROLOGUE

As everyone knows, the digital computer did not arrive from outer space in 1951. The story of the steady progression of machines from the Jacquard loom, the Babbage Analytic Engine, the Hollerith Electric Tabulating Machine, the IBM Electronic Multiplier and the ENIAC to present computer systems is familiar. In this volume, Professor Heinz Zemanek recounts the less known tale of pre-computer history in Central Europe.[3] The reader will discover that Jacquard had predecessors and that the 1890 Austrian Census used punched cards. This event had been made

PLENARY SESSIONS

The '76 NCC will include four special plenary sessions open to all conference attendees. Each will be held in the Grand Ballroom of the New York Hilton Hotel and will feature major presentations on issues of particular relevance to the computing field, and to concerned members of the business community and the general public.

Keynote Address
Monday, June 7
10:15 a.m.
J. Paul Lyet
Chairman of the Board
Sperry Rand Corporation

International Plenary Session
Monday, June 7
1:15 p m.
Chairman: Bob O. Evans
President
IBM System Communications Division
Session Participants
Professor A. S. Douglas, University of London
Dr. Anatoly A. Dorodnicin, Academician and Director of Computer Systems, USSR Academy of Sciences, Moscow
Dr. Heinz Zemanek, Director of IBM Laboratory, Vienna
Shiro Omata, President, Nippon Univac Kaisha, Ltd., Tokyo

The Computer Profession
Tuesday, June 8
1:15 p.m.
Chairman: Dr. Ruth M. Davis
Director, Institute for Computer Sciences and Technology
National Bureau of Standards
Featuring:
AFIPS Presidential Address
Dr. Anthony Ralston

Public Policy and Computers
Wednesday, June 9
1:15 p.m.
Chairman: Janice C. Lipsen
President, Counselors for Management, Inc.
Featured Address
To be Announced

CONFERENCE
AT A GLANCE

		1	2	3	4	5	6
		MONDAY AFTERNOON		TUESDAY MORNING		TUESDAY AFTERNOON	
		2:30 pm - 4:00 pm	4:15 pm - 5:45 pm	8:30 am - 10:00 am	10:15 am - 11:45 am	2:30 pm - 4:00 pm	4:15 pm - 5:45 pm
COMPUTERS AND PEOPLE	A **SOCIETAL CONCERNS** Sutton, NYH	A1-2 *Saul Padwo* WORLD ENVIRONMENT FOR DATA PROCESSING		● DATA SECURITY ●			
				Eldred C. Nelson A3 DATA SECURITY IN THE DOD	*Rein Turn* A4 DATA CRYPTOGRAPHY	*Naomi Seligman* A5 DATA SECURITY IN INDUSTRY	*Dennis K. Branstad* A6 SECURITY IN COMPUTER NETWORKS
	B **COMPUTER PROFESSION** Royal B, AM	B1 *Margaret Fox* 25 YEARS OF JOINT COMPUTER CONFERENCING	B2 *Walter Anderson* INFORMATION PROCESSING IN THE YEAR 2000	B3 *Anita J. Cochran* COMPUTING IN EUROPE	● PIONEER DAY ●		
					John G. Brainerd B4 ORIGINS OF ENIAC	*Herman Goldstine* B5 DEVELOPMENT OF ENIAC	*Richard E. Merwin* B6 POST-ENIAC TRANSFER OF TECHNOLOGY
	C **ISSUES IN COMPUTING** Imperial B, AM	C1 *Ronald A. Frank* ROLE AND OBLIGATIONS OF THE TRADE PRESS	C2 *H. W. Bomzer* DATA PROCESSING CAREER PATHS	● PUBLIC ACCESS TO COMPUTERS ●			
				Janet Kiehl C3 PUBLIC ATTITUDES TOWARD COMPUTERS	*Evelyn R. Murphy* C4 PERSONAL COMPUTERS	*David H. Ahl* C5-6 PUBLIC ACCESS TO COMPUTER POWER	
	D **APPLICATIONS SERVING PEOPLE** Grand Ballroom East, NYH	D1-2 *Genevieve Greenwald-Katz* COMPUTERS IN ARCHITECTURE		● CRIMINAL JUSTICE SYSTEMS ●			
				Bernice Pantell D3-4 THE ALEMEDA COUNTY LAW ENFORCEMENT SYSTEM OF THE FUTURE		*Thomas J. Madden* D5-6 CRIMINAL JUSTICE INFORMATION SYSTEMS AND USE OF CRIMINAL RECORDS	
SYSTEMS	E **COMPUTER SYSTEMS** Grand Ballroom West, NYH	●		COMPUTER SYSTEM DESIGN ●			
		John C. Davis E1-2 STORAGE SYSTEMS		*Nancy Betz* E3 INTERACTIVE SYSTEMS	*Stephen S. Yau* E4 COMPUTER SYSTEMS RELIABILITY AND MAINTAINABILITY	*Gerald Estrin* E5-6 MODULAR COMPUTER DESIGN	
	F **SYSTEMS MANAGEMENT** Mercury, NYH	● COMPUTER		SYSTEM MANAGEMENT AND PLANNING ●			
		John V. Soden F1-2 LONG-RANGE PLANNING FOR COMPUTER USAGE IN LARGE ORGANIZATIONS		*Edward O. Joslin* F3-4 ECONOMIC REQUIREMENTS AND WORKLOAD ANALYSIS: ESSENTIAL ELEMENTS OF SYSTEMS ANALYSIS		*David S. Alberts* F5 ECONOMICS OF SOFTWARE QUALITY ASSURANCE	*David S. Alberts* F6 FUTURE DIRECTIONS IN SOFTWARE QUAL. ASSUR.
	G **NETWORKING** Imperial A, AM	*Peter E. Jackson* G1 LEGAL & REGULATORY TRENDS IN COMPUTER COMMUNICATION	*Ira Cotton* G2 PROTOCOLS FOR COMPUTER NETWORKS	*Franklin F. Kuo* G3 PACKET RADIO & SATELLITE NETWORKS	*Robert E. Kahn* G4 PROGRESS IN PACKET NETWORK INTERCOMMUNICATION	*Louis Pouzin* G5 INTERACTIONS BETWEEN PRIVATE & PUBLIC DATA NETWORKS IN EUROPE	G6 (SEE "SECURITY IN COMPUTER NETWORKS" ABOVE.)
	H **BUSINESS AND INDUSTRY SYSTEMS** Georgian B, AM	H1 *Carol Johnson* ENHANCING LIBRARY SYSTEMS	H2 *Greg E. Mellen* AIR TRAFFIC CONTROL	● WORD PROCESSING & OFFICE AUTOMATION ●			
				Harvey L. Poppel H3 COMMUNICATIONS, COMPUTERS & WORD PROCESSING	*David Farber* H4 COMPUTERIZED MESSAGE SYSTEMS	*Howard L. Morgan* H5-6 WORD PROCESSING & OFFICE AUTOMATION	
SCIENCE AND TECHNOLOGY	I **COMPUTER & DATA BASE ARCHITECTURE** Georgian A, AM	I1 *Noah S. Prywes* IMPACT OF AUTO. OF SYST. DESIGN ON DATA BASE ARCHIT.	I2 *Liba Svobodova* COMPUTER STRUCTURE	● COMPUTER ARCHITECTURE ●			
				Yaohan Chu I3-4 HIGH LEVEL LANGUAGE COMPUTER ARCHITECTURE		*Anne M. Gulick* I5 MULTIPROCESSING	*Tomlinson Rauscher* I6 DEVEL. APPLIC. ORIENTED COMPUTER ARCHITECTURE
	J **SOFTWARE** Trianon, NYH	J1 *Margaret Butler* SOFTWARE SHARING	J2 *Alan G. Merten* TRANSFERABILITY OF APPLICATION PROGRAMS & DATA BASES	● SOFTWARE DESIGN & ENGINEERING ●			
				Edward Yourdon J3-4 STRUCTURED DESIGN		*Raymond T. Yeh* J5-6 SOFTWARE ENGINEERING — WHAT TO EXPECT IN THE NEXT DECADE	
	K **COMPUTER SCIENCE** Gramercy, NYH	K1-2 *Nathaniel Macon* COMPUTER ARITHMETIC AND NUMERICAL METHODS		K3 *James S. Ketchel* TECHNOLOGICAL FORECASTING	K4 *Joyce A. Amenta* SOFTWARE FOR SYSTEMS	K5-6 *Murray Turoff* IMPLEMENTATION OF COMPUTERIZED CONFERENCING SYSTEMS	
	L **APPLICATIONS OF COMPUTER SCIENCE** Royal A, AM	●		ARTIFICIAL INTELLIGENCE ●			
		Saul Amarel & Edward Feigenbaum L1-2 APPLIC. OF ARTIFICIAL INTELL. TO SCIENCE & MEDICINE		*Iris Kameny* L3-4 INFERENCE SYSTEMS AND SPEECH RECOGNITION AND UNDERSTANDING		*Marvin Minsky & Seymour Papert* L5 ARTIFICIAL INTELLIGENCE & EDUCATION	*Leonard Friedman* L6 THE PRESENT AND FUTURE OF MOBILE ROBOTS

NYH — NEW YORK HILTON
AM . AMERICANA

Figure 1—Conference at a Glance

CONFERENCE AT A GLANCE (continued)

		7	8	9	10	11	12	13	14
		WEDNESDAY MORNING		WEDNESDAY AFTERNOON		THURSDAY MORNING		THURSDAY AFTERNOON	
		8:30 am - 10:00 am	10:15 am - 11:45 am	2:30 pm - 4:00 pm	4:15 pm - 5:45 pm	8:30 am - 10:00 am	10:15 am - 11:45 am	2:30 pm - 4:00 pm	4:15 pm - 5:45 pm
COMPUTERS AND PEOPLE	A **SOCIETAL CONCERNS** Sutton, NYH	● PUBLIC POLICY ISSUES I ●				● PUBLIC POLICY ISSUES II ●			
		Frank D. De George A7 WELFARE PAYMENTS AND THE COMPUTER	*Marilyn E. Courtot* A8 DATA BANKS IN THE FEDERAL ESTABLISHMENT	*Anthony J. Patinella* A9 EFTS: THE POLICY QUESTIONS	*William R. Weber* A10 EFTS: IMPLEMENTATION PROBLEMS	*Neal Gregory* A11 DATA COMMUNICATION POLICY	*Francis Gregory* A12 PRIVACY: THE POLICY QUESTIONS	*John Salasin* A13-14 PRIVACY: THE PSYCHOLOGICAL & SOCIOLOGICAL IMPLICATIONS	
	B **COMPUTER PROFESSION** Royal B, AM	● COMPUTERS AND THE PHYSICALLY HANDICAPPED ●				B11 *Gopal K. Kapur* EXECUTIVE MANAGEMENT MUST BECOME INVOLVED	B12 *Roy F. Keller* CONCEPTS IN PROGRAMMING AND ADP INSTRUCTIONAL SYS.	B13 *Louise Etra* COMPUTER ART: NEW BREED OF ARTIST TECHNICIAN	B14 *Patsy Scala* ARTISTS & THEIR USE OF COMPUTERS
		Robert Gildea B7 COMPUTERIZED BRAILLE TRANSLATION	*Steven L. Jamison* B8 COMPUTERS AND SIGN LANGUAGE	*Harry G. Hedges* B9 COMMUNICATION AIDS FOR THE NON-ORAL	*Harry G. Hedges* B10 READING MACHINES FOR THE BLIND				
	C **ISSUES IN COMPUTING** Imperial B, AM	● INDUSTRY & UNIVERSITY RELATIONSHIPS ●				● SOFTWARE PRODUCTIVITY ●			
		Marshall C. Yovits C7 INDUSTRY NEEDS & VIEWS OF COMPUTER SCIENCE GRADUATES	*Marshall C. Yovits* C8 COMPUTER SCIENCE GRADUATES & INDUSTRY	*Marshall C. Yovits* C9-10 INDUSTRY AND UNIVERSITY — PROBLEMS AND SOLUTIONS		*Lloyd Baldwin* C11 PRODUCTIVITY PAYBACK FROM PACKAGED APPLICATION SOFT.	*Eugene I. Lowenthal* C12 HIGH LEVEL LANGUAGES FOR SOFTWARE DEVELOPMENT	*Larry A. Welke* C13-14 QUALITY AND PERFORMANCE MEASUREMENTS FOR SOFTWARE	
	D **APPLICATIONS SERVING PEOPLE** Grand Ballroom East, NYH	● MEDICINE AND HEALTH CARE I ●				● MEDICINE AND HEALTH CARE II ●			
		Thelma Estrin D7 COMPUTERS AND BIO-CYBERNETICS	*Robert S. Ledley* D8 COMPUTERIZED TOMOGRAPHY	*Richard Shepard* D9-10 BIOMEDICAL DATA BASES		*David J. Mishelevich* D11-12 TOWARD THE INTEGRATED HOSPITAL INFORMATION SYSTEMS		*Berrice J. Proctor* D13-14 PATTERN RECOGNITION IN CLINICAL MEDICINE	
SYSTEMS	E **COMPUTER SYSTEMS** Grand Ballroom West, NYH	● MICROPROCESSORS ●				● MINICOMPUTERS ●			
		Reg E. Kaenel E7-8 UNDERSTANDING AND USING MICROPROCESSORS		*Barry R. Borgerson* E9-10 MICROPROCESSOR SYSTEMS		*S. Ron Oliver* E11-12 MINIS VS. MAXIS		*Carol Brown* E13-14 USING MINIS IN LARGE AND SMALL BUSINESSES	
	F **SYSTEMS MANAGEMENT** Mercury, NYH	● COMPUTER SYSTEM PERFORMANCE & EVALUATION I ●				● COMPUTER SYSTEM PERFORMANCE & EVALUATION II ●			
		Philip J. Kiviat F7-8 COMPUTER PERFORMANCE MANAGEMENT		*Gerald Estrin* F9 MEASURES OF PERFORMANCE	*Stephen T. Swift* F10 PERFORMANCE INFORMATION SYSTEMS	*Jeffrey P. Buzen* F11 ISSUES IN COMPUTER SYSTEMS PERFORMANCE	*Jack Moshman* F12 PROBAB. MODELS FOR PREDICT. SYS. THROUGHPUT	*William E. Perry* F13-14 COMPUTER SYSTEMS AUDITABILITY AND CONTROL	
	G **NETWORKING** Imperial A, AM	*Teresa O. Green* G7 USE OF NETWORKS IN SCIENCE & EDUCATION	*Walter A. Sedelow* G8 NETWORK ARCHITECTURE	*Stephen R. Kimbleton* G9-10 NETWORK OPERATING SYSTEMS		*Susan Poh* G11-12 NETWORK MEASUREMENTS		*Thomas Pyke, Jr.* G13-14 NETWORK ACCESS TECHNIQUES	
	H **BUSINESS AND INDUSTRY SYSTEMS** Georgian B, AM	● COMPUTER-ASSISTED MANUFACTURING ●				● COMPUTER-CONTROLLED PUBLICATION ●			
		Thomas L. Boardman H7-8 COMPUTER-AIDED MANUFACTURING AND DESIGN		*W. Barkley Fritz* H9-10 COMPUTERS IN THE SHIPBUILDING INDUSTRY		*Norman W. Scharpf* H11-12 COMPUTERS IN TYPESETTING		*Dennis R. Neary* H13-14 INTEGRATION OF MICROFILM AND COMPUTER	
SCIENCE AND TECHNOLOGY	I **COMPUTER & DATA BASE ARCHITECTURE** Georgian A, AM	● DATA BASE ARCHITECTURE ●				● DEVELOPING DATA BASE SYSTEMS ●			
		Etelle Grinoch I7 DATA BASE STRUCTURE	*Susan Brewer* I8 RELATIONAL DATA BASES	*John L. Berg* I9-10 DATA BASE DECISIONS		*Fredric C. Gey* I11-12 SOCIO-ECONOMIC FACTORS IN DATA BASE MANAGEMENT		*Patricia Berglund* I13 MANAGEMENT INFORMATION SYSTEMS	*Norman F. Hirst* I14 AN INTRODUCTION TO MUMPS
	J **SOFTWARE** Trianon, NYH	● PROGRAMMMING LANGUAGES ●				J11-12 *Edward Miller* AUTOMATED SOFTWARE TESTING AND EVALUATION		J13-14 *Paul Schneck* WHY (OR IF) FORTRAN WILL SURVIVE	
		Herbert Maisel J7-8 PROGRAMMING LANGUAGE DESIGN		*Paul Oliver* J9-10 PROGRAMMING LANGUAGE STANDARDS: SUCCESSES AND DISAPPOINTMENTS					
	K **COMPUTER SCIENCE** Gramercy, NYH	K7-8 *Enrique Ruspini* APPROXIMATE REASONING AND APPROXIMATE ALGORITHMS		K9 *John McLeod* ENERGY MODELS	K10 *Roger M. Firestone* COMPUTING PHYSICS	● MATHEMATICAL PROGRAMMING ●			
						Patricia J. Eberlein K11 ALGORITHMS FOR UNCERTAIN FORMS	*Jean-Paul Jacob* K12 SIMULATION VS. MATHEMATICAL PROGRAMMING	*Darwin Klingman* K13-14 OPTIMIZATION OVER GRAPHS AND NETWORKS	
	L **APPLICATIONS OF COMPUTER SCIENCE** Royal A, AM	● COMPUTER GRAPHICS ●				● COMPUTER STUDIES IN THE HUMANITIES ●			
		Stephen Levine L7 SOUND — ANOTHER DIMENSION IN COMPUTER GRAPHICS	*Joseph Scala* L8 TEACHING THROUGH COMPUTER GRAPHICS	*Richard Speer* L9 COMPUTER GENERATED FILMS	*Jackie Potts* L10 COMPUTER GRAPHICS SOFTWARE	*Joseph Raben* L11 COMPUTERS IN MUSICAL & HISTORICAL RESEARCH	*Naomi Sager* L12 COMPUTATIONAL LINGUISTICS	*A. K. Joshi and C. Rieger* L13-14 NATURAL LANGUAGE PROCESSING	

Figure 1—(Continued)

possible by a remarkable, adopted Austrian engineer, Otto Schaeffler, whose capsule biography is delightful and fascinating. Another fascinating story, and not as well known as it should be, is the history of early computers in Europe. Richard Williams recounts this history with authority and rare candor. Machines were built in Germany, Holland, France, Scandinavia and Great Britain. It is very interesting to learn that the only German commercial computer development was by Konrad Zuse based on the control mechanisms for V1s and V2s. This paper contains many very interesting notes and remarks. Williams pays tribute to the foresight of Eckert and Mauchly in recognizing the business potential of computers. In England, influenced by Professor Hartree's remarks that computers would never be used for business purposes because there would not be enough scientists to run them, commercial use was delayed. During the 1976 Conference, we pay tribute to the development of the ENIAC at the Moore School of Electrical Engineering at the University of Pennsylvania in the Pioneer Day activities, organized this year by Dr. Harvey L. Garner, the director of the Moore School. It is a matter of regret, for the Program Chairman, that papers on the topics to be discussed on Pioneer Day are not available to be included in this volume. Personal accounts of the development of our profession are a valuable part of our heritage.

COMPUTERS AND PEOPLE

This area is introduced by the seminal paper by Professor A. S. Douglas who quite properly points out the necessity of separating Privacy, Confidentiality and Security. This, Professor Douglas does with decisive clarity in his discussion on privacy and data protection procedures in the U.K. Privacy, data security and electronic funds transfer are major societal concerns. Social impacts and technological aspects of these topics are addressed by a group of papers which are included in this volume. These papers are only a sample of the content of the program which expands these topics with panel discussions on policy questions and implementation problems. The program also includes panels on world environment for data processing, welfare payments and the computer, data banks in the federal establishment and data communication policy. The papers published here provide the background for these panel discussions.

The track on the Computer Profession contains papers on education and training, exciting developments in the use of computers to generate Art, and some very intriguing developments in the field of computers and the physically handicapped. The computer has great promise for providing educational and occupational opportunity for the physically handicapped. Clearly there is much good work being done, but equally clearly much more research, development and engineering is needed. A central difficulty is apparently the need both to understand the requirements of the handicapped and the levels of acceptability in equipment developed for them. The paper on a hand-held calculator for the blind reveals what can be accomplished by a combination of imagination and technical skill.

David Ahl organized a very interesting full day on the subject of Public Access to Computers. The three papers on that topic published in this volume give only a glimpse of what is happening. A few years ago the concepts of amateur computing, home constructions of computers and universal access to computer power, would have been dismissed as ludicrous. Today there are thousands of home built and operated computers.

In the track on applications serving people, there are papers in medicine and health care, criminal justice systems and architecture. The variety and versatility of the applications which are described is surprising to one who has been immersed in the development of computer systems. The computer technologist finds it hard to recognize "his" computer in the description of these applications. While we always said that truly remarkable applications would evolve, did we really suspect that they would be so remarkable? One task for the future, in which we must all participate, is to retain a common language through which we can communicate with one another. No Tower of Babel must be allowed to arise.

SYSTEMS

If there is one area in which the difference between the first and the forty-fifth conference is marked, it is in the area of systems. The program for the 1976 conference has 38 sessions in which the participants will examine and explore computer system design, microprocessors, minicomputers, computer system management and planning, computer system performance and evaluation, computer networking in the United States and Europe, word processing and office automation, computer-assisted manufacturing and computer-controlled publication. The evolution of the systems approach and the development of systems has occurred so naturally, that it is hard to recall now how we thought about such things in 1951. The preoccupation was in making the calculators and computers work, and little, if any, thought was spared for system design. Learning to think in terms of systems and implementing system designs is one of the great achievements of the past quarter of a century. Again, only a sampling of what is discussed in the program is printed in this volume, but that sample presents an interesting view of the current state of systems.

The track on Computer Systems begins with a perceptive paper by Margaret Butler. Four generations of computer technology ranged from the vacuum tube (1946) through the transistor (1959) and the integrated circuit (1965) to large-scale intergration (1971). During the same time frame, memory was evolving from the mercury delay line and the electrostatic tube through magnetic cores and plated wires to semi-conductor memories. The paper by Mrs. Butler is an interesting discussion of the development of large-scale computer systems. Another very interesting paper is the 1975 evaluation at the Control Data Corporation of the East German RYAD 1040 system. On the basis of the performance tests reported, there is an apparent lag behind Western technology in processors, memory, and peripheral equipment, but workmanship and reliability were, in general, very good. At the other end of the computer system spectrum is the application of a microprocessor to the handling of bowling scores. Microprocessor systems are being developed which combine low cost, small size and modest power requirements with high operating speed, a high density of logic and flexibility in configuration. It is now hard, if not impossible, to distinguish between a micro and a small mini on the basis of performance. The management of systems now requires greater technological skill as well as an understanding of the basic economic aspects. Quality control of software, reliability, and the evaluation and prediction of performance of computer systems are major management concerns. None of these subjects is simple, but the effective manager must learn to understand them. Improved performance and increased productivity are necessary for profitability.

A special feature of this year's conference is the particular attention paid to Networking. In addition to the ten sessions so ably organized by Ira Cotton and Franklin Kuo, there are also a demonstration of a commercial network and a professional development seminar on Networking. Taken together, these events present an integrated approach to Networking which should be useful to everyone with an interest in the topic. Fifteen papers on Networking and related subjects are printed in this volume. Included among these papers is an extensive review by Frank Martin of the FCC dockets which affect computer communication. Decisions already made by the Commission appear to indicate, if only vaguely, how they will distinguish between communications and data processing. But the issues are not settled and many hearings and court cases will transpire before the issues are settled. A proposed international packet-switched network protocol is discussed in a paper by authors from four countries (Rybcznski, et. al) and Louis Pouzin presents a somewhat different point of view. Network access techniques, network architecture, network design problems, and the measurement of network performance are also treated by papers in this volume. From a laboratory curiosity in 1972 to a commercially available capability in 1976 is an amazingly swift development in which many individuals participated and contributed.

In the business and industry systems, which are described in this volume, the importance and value of computer based systems is clearly indicated. Work processing and office automation, computer-controlled publication, computer-aided design, computers in shipbuilding, air traffic control, and the expanding role of computers in libraries are described in this volume. They are not, themselves, the totality of applications of computers in business and industry but they do represent an impressive sample of such applications. In reviewing these developments, one becomes acutely aware of the maturing of the computer from a scientific toy to a business necessity.

SCIENCE AND TECHNOLOGY

In the area of science and technology, we are in the more traditional realm of the joint computer conference. It is also the area in which one would look for the insights into what the future might bring. If the papers received for this conference are any indication, we are now in a transition period in which older ideas are being consolidated and new ideas have not yet emerged. In computer architecture, there is still much interest in pipeline computers, multiprocessors and in parallel processing. What seems to be lacking is the corresponding system software development. The concept of application oriented computer architecture is intriguing as is the idea of coupling small computers for performance enhancement. Undoubtedly the world-wide economic conditions of the past few years have had a dampening effect. The great interest in data base is reflected in the number of papers on that subject. Relational data bases remain interesting but there are still questions concerning their practicability and efficiency. Both performance and user acceptibility are important considerations. The papers on data base structures and on management information systems show that there are important problems to be solved although considerable progress has been made. Error detection in data bases and integrity aspects of shared data bases are topics that will continue to merit discussion.

In the computer science and application of computer science tracks, an interesting collection of papers appear. There are two fine papers on networks in the Operations Research sense of that term. Computer arithmetic and numerical methods are addressed as well as algorithms for uncertain forms. Cynthia Solomon's case study of a child doing turtle graphics in LOGO was fun to read. The two papers on fuzzy sets by James O. Bezdek and Richard A. LeFaivre, which are part of the session on Approximate Reasoning, seem to represent a future direction for the analysis of computer systems.

REMARKS

Each conference is, to paraphrase Emerson's remark, the lengthening shadow of its participants and attendees. The discussions and debates during the sessions are both the real substance and the ephemeral part of the Conference. In the past, recordings of the sessions have not seemed to capture the vital spirit of the meetings. Thus we are left with the Proceedings as the lasting record of the Conference. For the errors of omission and commission, the Program Chairman must accept responsibility. For what is fine and worthwhile in this volume, credit must be given to the authors and to the reviewers, to the members of the Program Committee and the Program Advisory Committee, and to the Steering Committee. Everyone has contributed in a significant way. There are five individuals without whose dedicated effort, repeatedly given over long hours, this volume would not have been finished. It is with most grateful appreciation that I mention Carl Hammer, who did more than all of us and provided leadership and inspiration as well, Norm Moraff, Lee Danner, Anita Cochran and Nelle Morgan.

REFERENCES

1. Weiner, Norbert, *The Human Use of Human Beings*, Houghton Mifflin Co., New York, 1950.
2. Grosch, Herbert R. J., "Conference Maketh a ready man or, twenty-five years in the better joints," *AFIPS Conference Proceedings* Vol. 45, 1976. Figure 1 of this paper is the table of contents of the Proceedings of the First Conference.
3. References to papers which appear in this volume will not be given. The reader is referred to the Contents or the Author Index for any individual paper.

Conference maketh a ready man
Or, twenty-five years in the better joints

by HERBERT R. J. GROSCH
Consulting Editor, COMPUTERWORLD
and *Vice President, Association for Computing Machinery*
Sunnyvale, California

ABSTRACT

The author rehearses, with much pleasure, the origins, physical circumstances, personalities, exhibits and papers of the Joint and National Computer Conferences, from December 1951 to the present meeting.

The Joints are no more, at least in name—but long live the NCCs! I suffered the agonies of grim Philadelphia at the very first Joint, not yet named "Eastern." I enjoyed the dubious sunshine of Los Angeles at the first Western, not yet named "Joint." I was on the JCC Board twenty years later, when, hoping to retrieve the big exhibitors of the Sixties, our Industry Advisory Panel told us

> "The time is out of Joint;"

and we replied, each one of us,

> "O cursed spite,
> That ever I was born to set it right"!

And we coined the name, National Computer Conference, to mark the creation of something new, yet old: still joint, but of all the AFIPS societies; no longer Joint, or Western, or Eastern, or Spring, or Fall. And, Disneyland coupon book in hand, I scurried off to the last of the old semiannuals: Fall, 1972.

So I may reasonably claim, I think, to have been as close to the Joints as any man could be, and to all the NCCs so far. (See Table I.)

There are 45 conferences, counting this one, and they stretch over a full quarter century. I attended 35, and would have gone to more except for a long sojourn in Western Europe in the early Sixties.

Recent comers to our trade can hardly imagine the novelty of a computer conference in 1951. The first production computer, a UNIVAC I, had just been delivered to the Census. The most powerful IBM machine in production was the ineffable Card-Programmed Calculator. The Association for Computing Machinery was only four years old, and was years away from its first formal publication series. The IEEE did not yet exist, and the two societies which

TABLE I

Number	*Type*	*Opened*	*Days*	*Place*
1	JCC	1951 Dec. 10	3	Philadelphia
2	JCC	1952 Dec. 10	3	New York
3	WCC	1953 Feb. 4	5	Los Angeles
4	EJCC	1953 Dec. 8	5	Washington
5	WCC	1954 Feb. 11	2	Los Angeles
6	EJCC	1954 Dec. 8	3	Philadelphia
7	WJCC	1955 Mar.1	3	Los Angeles
8	EJCC	1955 Nov. 7	3	Boston
9	WJCC	1956 Feb. 8	3	San Francisco
10	EJCC	1956 Dec. 10	3	New York
11	WJCC	1957 Feb. 26	3	Los Angeles
12	EJCC	1957 Dec. 9	5	Washington
13	WJCC	1958 May 6	3	Los Angeles
14	EJCC	1958 Dec. 3	3	Philadelphia
15	WJCC	1959 Mar. 3	3	San Francisco
16	EJCC	1959 Dec. 1	3	Boston
17	WJCC	1960 May 3	3	San Francisco
18	EJCC	1960 Dec. 13	3	New York
19	WJCC	1961 May 9	3	Los Angeles
20	EJCC	1961 Dec. 12	3	Washington
21	SJCC	1962 May 1	3	San Francisco
22	FJCC	1962 Dec. 4	3	Philadelphia
23	SJCC	1963 May 28	3	Detroit
24	FJCC	1963 Nov. 12	3	Las Vegas
25	SJCC	1964 Apr. 21	3	Washington
26	FJCC	1964 Oct. 27	3	San Francisco
27	FJCC	1965 Nov. 30	2	Las Vegas
28	SJCC	1966 Apr. 26	3	Boston
29	FJCC	1966 Nov. 7	4	San Francisco
30	SJCC	1967 Apr. 18	3	Atlantic City
31	FJCC	1967 Nov. 14	3	Anaheim
32	SJCC	1968 Apr. 30	3	Atlantic City
33	FJCC	1968 Dec. 9	3	San Francisco
34	SJCC	1969 May 14	3	Boston
35	FJCC	1969 Nov. 18	3	Las Vegas
36	SJCC	1970 May 5	3	Atlantic City
37	FJCC	1970 Nov. 17	3	Houston
38	SJCC	1971 May 18	3	Atlantic City
39	FJCC	1971 Nov. 16	3	Las Vegas
40	SJCC	1972 May 16	3	Atlantic City
41	FJCC	1972 Dec. 5	3	Anaheim
42	NCC	1973 June 4	5	New York
43	NCC	1974 May 6	5	Chicago
44	NCC	1975 May 19	5	Anaheim
45	NCC	1976 June 7	4	New York

Key: CC=Computer Conference, E=Eastern, F=Fall, N=National, S=Spring, W=Western.

later merged to form it did not themselves care much for the miniscule computer activities of the time. The Institute of Radio Engineers had had an Electronic Computer Committee since early 1948, for which I had helped produce a bibliography (a few dozen entries) in 1949, under the direction of Bob Serrell. The American Institute of Electrical Engineers had a Committee on Computing Devices—and analog "devices" were also hot stuff in the Forties and early Fifties.

These two committees in turn appointed, in early 1951, a *joint* committee to arrange a conference. That first JCC met in Philadelphia, the world center of electronic computer activity at the time. The Moore School, the Eckert-Mauchly division of Remington Rand, the proximity to Aberdeen Proving Ground (the largest computer center of that day, with a huge differential analyzer, punched card machines, IBM and Bell Labs relay calculators, and of course ENIAC) and the relative closeness of Washington, all contributed to this judgment. Washington would have been suitable also, and was indeed the site of the third eastern meeting two years later. It was the source of most funding of one-off machines, the purchaser of most production computers, the nexus of enthusiasms for the Defense Calculator, IBM's yet-unannounced scientific computer (called Type 701 from 1952 on). Another main factor in the choice of Philadelphia was that attendees could actually see, feel, smell the equipment—and that excitement led to the exhibit idea, and to the National Computer Conference as we see it today.

In February 1952, delighted with the unexpectedly large attendance (almost a thousand) the committee published with AIEE help the first Joint proceedings. Figure 1 gives the table of contents; note the early appearance of British participants (both Cambridge and Manchester); note Jean Felker's paper on using transistors, then less than three years old; note that monuments such as Howard Aiken's MARK III, Sam Alexander's SEAC, and Jay Forrester's WHIRLWIND I were described. And note the first big drum-dominated 1100-series machine, already at work on cryptographic problems: ". . . the user is not free to talk about his classified applications."

A few last words about origins: there had been a jointly-sponsored meeting on electron tubes for computers in 1950 (Atlantic City, thus setting a horrid precedent), and except perhaps for an early lack of perspective that somewhat smaller gathering could have been labeled the first Joint Conference. And ACM, although interested more in hardware in 1951 than in later years, only "cooperated" in organizing the Philadelphia sessions. The title page of the very first meeting says "Joint AIEE-IRE"; of the second, "Joint AIEE-IRE-ACM"!

TABLE OF CONTENTS

	Page
Keynote Address, W. H. MacWilliams, Jr.	5
The UNIVAC System, J. Presper Eckert, Jr., James R. Weiner, H. Frazer Welsh, Herbert F. Mitchell	6
Discussion	15
Performance of the Census UNIVAC System, J. L. McPherson, S. N. Alexander	16
Discussion	20
The Burroughs Laboratory Computer, G. G. Hoberg	22
Discussion	29
IBM Card-Programmed Calculator, J. W. Sheldon, Liston Tatum	30
Discussion	36
The ORDVAC, R. E. Meagher, J. P. Nash	37
Introductory Remarks, Dr. Herman Goldstine	38
Discussion	42
Design Features of the ERA 1101 Computer, F. C. Mullaney	43
Discussion	49
The Operation and Logic of the MARK III Electronic Calculator in View of Operating Experience, Glen E. Poorte	50
Discussion	56
The University of Manchester Computing Machine, F. C. Williams, T. Kilburn	57
The Design, Construction, and Performance of a Large-Scale General-Purpose Digital Computer, B. W. Pollard	62
Joint Discussion	69
The Whirlwind I Computer, R. R. Everett	70
Evaluation of the Engineering Aspects of Whirlwind I, Norman H. Taylor	75
Joint Discussion	78
The EDSAC Computer, M. V. Wilkes	79
Discussion	83
The National Bureau of Standards Eastern Automatic Computer (SEAC), S. N. Alexander	84
Engineering Experience with the SEAC, Ralph J. Slutz	90
Joint Discussion	93
Computing Machines in Aircraft Engineering, Charles R. Strang	94
A Review of the Bell Laboratories' Digital Computer Developments, E. G. Andrews	101
The Transistor as a Digital Computer Component, J. H. Felker	105
Discussion	109
Digital Computers—Present and Future Trends, J. W. Forrester	109
Discussion	113

Figure 1

I remember rather faintly that hotel arrangements were pretty grim, and that social events were limited to privately-arranged tours to the UNIVAC, to Burroughs, and to the Moore School. I remember vividly buttonholing everybody I knew to tell them how great IBM's new machine was going to be—I was fresh from several months in far Poughkeepsie! I remember John Bennett saying there would be an Australian meeting the next year (and there was!). I remember there were a few senior women professionals present, but none on the committees or program. And I remember being impressed that Charlie Strang, a vice president of Douglas Aircraft, would come all the way from Santa Monica to gray Market Street to tell us a user's story—the only such paper. The weather? I don't remember. Did I bring my wife? No. Did I enjoy it? Well, yes; great stuff about several important new ventures, and hot news from England. But I didn't get to say anything myself!

In those days we read MTAC, "Mathematical Tables and other Aids to Computation," a *very* curious National Research Council publication. It still exists, in a much more esoteric format. Along with errata in printed tables, book reviews of new numerical analysis texts, and the like, the editors reviewed articles on computers and computing, described new machines and acceptance tests, and told about conferences and seminars. In 1952 they gave the program of the Philadelphia meeting, and in 1953 published a six-page review of the Proceedings, by Ed Cannon of the Bureau of Standards. This was the first appearance in the nonengineering literature of the many, many references over the decades, to our formidable conferences.

The second year the show really got on the road, in a manner of speaking. First of all, there were exhibits—and from that, in late 1952, we never receded. Then it traveled: met in New York. And clouds not much larger than a man's hand were visible: a 25 percent gain in attendees, and a Western Liaison subcommittee. Aided by the proximity to Galactic Headquarters at Madison and 57th, the various committees tapped IBM for personnel and other support: I had expected to be involved, although then based in Washington, but was extruded from the Body Economic for general recalcitrance only a few weeks before. This probably accounts for my rather specialized impression of JCC2: Jay Forrester offered me a job at Whirlwind!

Margaret Fox of NBS did most of the program, and there was a paper on SEAC offline input-output gear by Ruth Haueter: first major feminine influence. The whole program, in fact, revolved around peripherals, from magnetic wire(!) and tape equipment to Kimball tags. Paper tape, punched cards, photographic techniques (including a read-only disk), and the ubiquitous line printer all were discussed. And of great interest to me, of course, the first major papers on the IBM 701.

Don Davies, today the top man at the National Physical Laboratory, was a new but welcome visitor. General Electric made an appearance, threatening to build a nonimpact printer. And artificial intelligence, the advisability of standards, and economic modeling were all mentioned—just like 1976!

The scene now shifts to that hotbed of technical computing, Los Angeles. The tin airplane was flourishing, missiles were at least conceivable, and spies had been sent to the East and had reported, notably Harry Huskey and Dick Canning. The joint committee decided to try a western conference. And because the enormous later development of componentry, of peripherals and of systems had not yet flowered, most of the papers concerned applications (Figure 2). Canning had just come down from Mugu, McCann and Morton were pushing hardware at Cal Tech and Berkeley, the Northrop offshoots had incorporated (CRC, later to be a part of National Cash). There were exhibits: I particularly relish the memory of a Telecomputing point plotter that counted the lines on graph paper.

PROCEEDINGS of the WESTERN COMPUTER CONFERENCE

Table of Contents

KEYNOTE AND LUNCHEON ADDRESSES

SESSION III

SESSION I

SESSION II

SESSION IV

SESSION V

Figure 2

I also relish, perhaps comfortably in view of the dominance of digital ideas today, the comment I made to Arthur Vance, a prominent RCA analog man, that effort should preferably be spent on numerical analysis, and less on stringing "900 integrators on the end of one piece of wire. (Laughter and applause.)" *Plus ça change* . . .

Now that there had been a Western, and a reasonably successful one, the senior conference had to be relabeled. So from that point on, we had *Eastern* Joint Computer Conferences—nine of them, through 1961. And, as the third eastern conference was called "Eastern," so the third one on the Coast would be called "Joint," in 1955.

That first EJCC was held in Washington, still in early winter. I gave the after-luncheon speech, which does *not* read too well almost 23 years later. But the papers do, notably the first conference report on magnetic core memory, first mentions of life insurance "Electronic Data-Processing" and of numerical weather prediction. One of the very earliest interpreters, the Los Alamos SHACO (Short Hand Coding), was described, and discussion of open shop versus closed shop appeared. I remember the former: Allan Benson had worked up a three-address floating point package for the very first 701. And as for the latter, why, I had a great closed shop in Cincinnati and couldn't see why anyone would want the opposite: "it requires more tactful people," said the Los Alemite. What I wanted was core instead of electrostatic memory; *definitely* not tact!

This Washington meeting was keynoted by Howard Engstrom, and his presence was an early link to the National Security Agency. The next Western, two months later, saw some of the bedroom conferences that led to SHARE later that year (1954), and NSA, super-secret though it was, became a charter member.

The Washington meeting was perhaps the last in which *everybody* discussed reliability. There would be dour jokes in the trade for many conferences to come about specific hardware problems: air conditioning, head crashes, and so on, up into modern times. But increasingly there would be concern for software problems; the hardware was working.

There were exhibits, but in the small Statler environment; the day of the giant hall and the gorgeous booth bird had not dawned.

Well, there was one more Western Computer Conference to go, at the Ambassador in Los Angeles—I remember I ran my first major recruiting suite. The exhibit list was getting longer, and space was tight. There was heavy emphasis on control applications: machine tools, chemical processes, and feedback of management information—the earliest MIS discussion group. I was on it, boasting about Stan Rothman's machine shop scheduling work in General Electric.

Sybil Rock, Monty Phister, Herb Mitchell were all flourishing; Harry Huskey had gotten to UCLA, Louis Ridenour was playing his games at International Telemeter. It was a small but golden time.

That December, the Easterners turned back to Philadelphia, and I was program chairman. This was still the era of single sessions, mind you, and of one-man program committees. I chose small digital computers for a subject; tongue-in-cheek, no doubt, Charlie Adams did a keynote that mentioned a fictional giant machine: "Officially the giant brain was the SOCIAC, but . . . around the office it was known as Herbie." Alan Perlis, of all people, did the survey paper. How have the leaves fallen: the IBM 650, the Marchant Miniac, the Alwac! I read the other day there were "several hundred" minicomputer and microprocessor types in current production. In 1954 there were nine. Most were decimal, none were transistorized; one had an early cassette ("magnetic tape capsule").

Software? Not much, although Stan Gill had come over and was sharing his experiences. One 650 customer described an "automatic coding" technique; today we would call it an optimizing interpreter. And in the information retrieval field, one of the earliest appearances of the team of Perry, Berry and Kent, then at Battelle, was also a near-first for that subject at a Joint.

By this time the three sponsoring groups saw they had a Good Thing going, and formalized the joint committee's structure, prescribed the steering committees for the two annual conferences, and set up financial procedures (the surplus was divided equally, and each conference started from scratch, with volunteer workers and a small loan "from each of the sponsors"). Clearly, a machine had started to grind.

The seventh one, the first to be called Western *Joint*, I missed. ACM records say it was at the Statler in Los Angeles, and AFIPS records show 1500 registrants, double the 1953 startup number. There were trips and exhibits, and a big publicity thrust. Don Pendery of IBM was on a panel about common languages. On the other hand, there *was* a lot of analog material on the program, and in the Spring of 1955 I was helping persuade General Electric Syracuse, in the person of the famous W. R. G. Baker, to start its own computer adventure. So, although I had lots of travel money, and a continuing need to recruit—one of the major activities at all early Joints—I passed it up.

The papers list includes Charlie De Carlo and Willis Ware, Newell on chess machines, and Bob Johnson's doctoral thesis. Must have been a good meeting!

By this time SHARE was official, *Fortune* cared about computers, and IBM was delivering 704s. The president of Burroughs, no less, came to Boston for the 1955 EJCC and talked about computers as management tools: not too sensibly, as I remember it, but it flattered us all nevertheless. Indeed, the conference was quite strongly DP oriented; even Tony Oettinger, then a humble instructor at Harvard, did a piece, as did Bob Gregory. There were review papers on information retrieval and on data communications networks. Networks, yet! And I did my first-ever paper on standards; fortunately, it had been forgotten before I was interviewed for the NBS job ten years later! It was followed by a sound—soundly pessimistic—review of magnetic tape standardization problems by Ampex and ElectroData and telephone and government people.

Finally, Jay Forrester in a conference summary referred to computer toys and computer kits and electronic surplus gear, a prevision of the myriad hobby enterprises of LSI 1976.

Space of course doesn't permit a review of every Joint and National; moreover, after the first ten or twelve, media developments make information and impressions more easily available. Many libraries have the later conference proceedings; many libraries and individuals have access to the JCC issues of *Datamation,* which began covering the Joints in 1957. What I therefore will do from 1956 on is to skip along, recalling high points, personal or professional, and relating them to the rapidly developing world outside the three societies and their enterprises.

The 1956 WJCC was held at the Fairmont in San Francisco, beginning a love affair with that town which lasted until the exhibits finally outgrew available space in 1968. Shortly thereafter I moved to Phoenix to help GE enter the field, and was put on the Western Conference Committee. The 1956 EJCC reworked the organizational structure behind the conferences (they were growing at incredible speed), and created a National Joint Computer Committee. It was the existence of these initials, NJCC, for so many years thereafter that militated against adoption of NJCC in 1972, as the initials of the new once-a-year conference and exhibit. The letters "NCC" were chosen instead.

The 1956 conferences, taken together, were dedicated to great projects: BIZMAC, the DATAMATIC 1000, the Univac LARC, and the IBM STRETCH. The first IBM 705 was delivered, tubes and all, to Jack Jones in Atlanta: first time I'd heard of him. The 709, tubes and all, wabbled onto the scene. But in the forefront, in single copies but vastly significant, were the first powerful transistor machines: the MIT TX-0, with 65K words of core, and the Transac S-1000. Oh, and IBM brought out the first disk machine: RAMAC. I remember the latter not so much for its novel appearance but for the fact that the IBM engineer who described the machine actually quoted the rental!

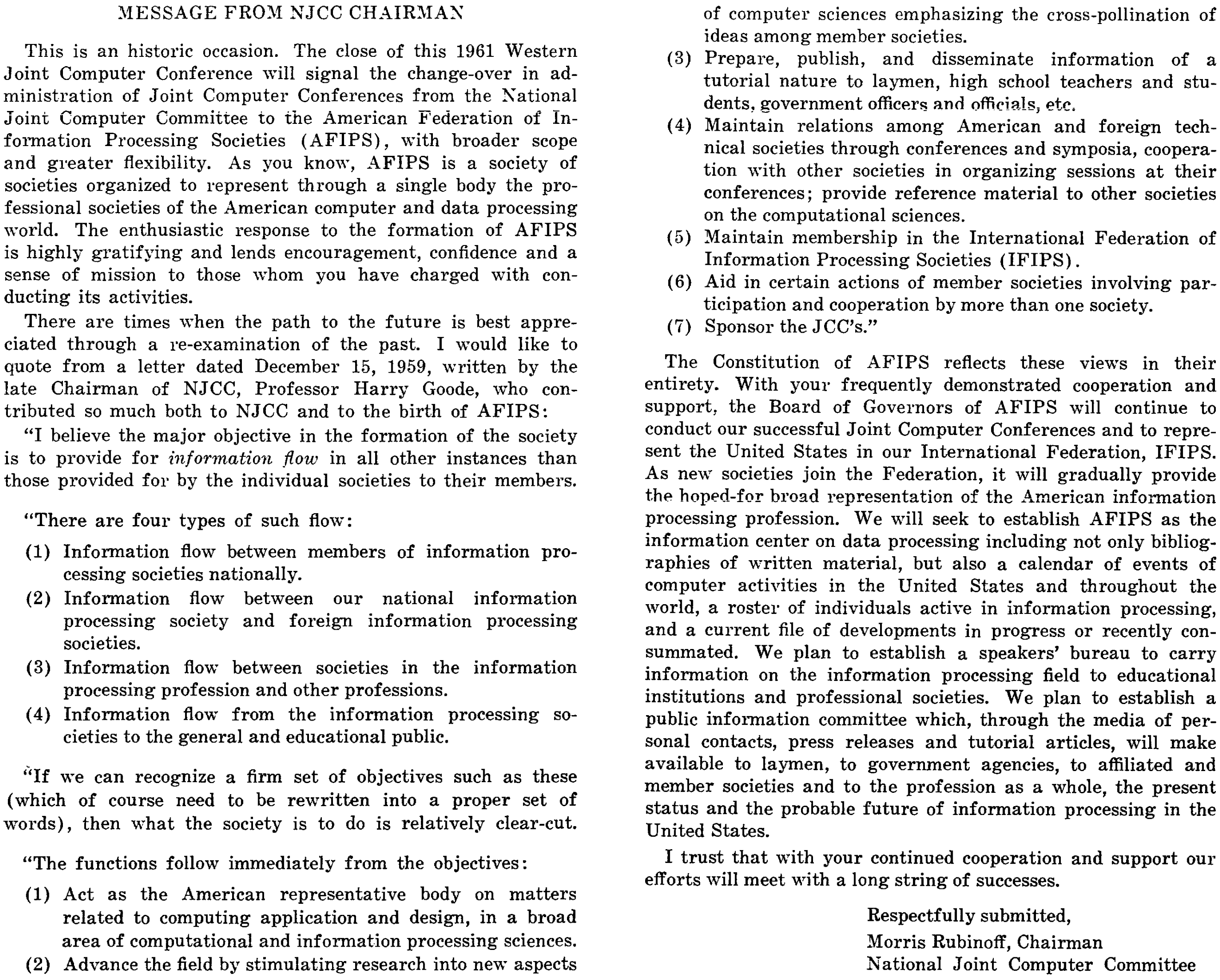
For the 1957 Washington conference I still have my registration receipt: $4.00. Those were simpler times: single sessions, straightforward entries like the Bendix G-15, open scandals like SAGE. For the first time, a fourth organization, the National Simulation Council, shoved its tiny nose into the tent.

In 1958, social implications: a great panel which I remember vividly, with a famous Yale professor, a great union man, and IBM's Cuthbert Hurd (severally identified in the Proceedings as "Nonmember AIEE"!) taking an early look at automation. For the first time, multiple sessions. And for the first time on any continent, the Bull Gamma 60, with Phillippe Dreyfus; I helped him a little. At the session that winter, in Philadelphia again, Heinz Gumin, now the top Siemens man in computers, came over for the first time, and described the 2002. I was more interested at the time, not realizing that I would soon be moving to Europe, in the paper on microprogramming by Maurice Wilkes, by now almost a fixture at the eastern meetings. Also, having served my time in Phoenix, I noted but did not attend the paper on the magnetic-character check handler, ex-ERMA.

Cal Mooers was in great form in San Francisco: information retrieval. And I did the ladies program. Also Charlie Asmus, for many years to follow the key figure in staging the ever-growing exhibits, began major participation at the meeting.

The next vivid memory I have is of EJCC 1960. It was held at the Manhattan Center and the New Yorker Hotel, just before Christmas. And the night before, December 12, it snowed. And snowed. And snowed! Most of the exhibitor trucks, and most of the attendees, missed the first day entirely. I lived in New York that year, and arrived promptly, by subway. It was this near-catastrophe, modulated of course by the several-year lead times now necessitated by the size of the Joints, that led to the changed pattern of the Sixties. From 1951 on, the eastern conferences had been held in December, and in the East the weather was frequently awful. The western conferences had drifted from February, too soon after the EJCCs, to May, when the weather was great everywhere. So in 1962 the old terms "Eastern" and "Western" were abandoned, the conferences were switched, and from that time to the end of the two-a-year Joints, the eastern meeting was held in May or so, and called "Spring," and the western one was held in November, always pleasant in the West, and called "Fall." In order to effect a transition without two meetings in a row on one side of the country or the other, the 1963 SJCC was held in Detroit; I came back from Europe to attend, partly because of the novelty, partly because, living overseas, I had missed three or four Joints in a row, and the friends I saw at them. That was the midpoint conference, in many ways: Number 23. NCC 76 is Number 45.

The next dislocation in the series was a simpler one: there was no Spring Joint in 1965. New York was the site that year of IFIP 65, the second of the triennial international meetings. The zeroth, pre-Federation, had been held at UNESCO in Paris in 1959, and the first, in Munich in 1962. So, preparing to sponsor the 1965 sessions and show, AFIPS gave up one Joint.



MESSAGE FROM NJCC CHAIRMAN

This is an historic occasion. The close of this 1961 Western Joint Computer Conference will signal the change-over in administration of Joint Computer Conferences from the National Joint Computer Committee to the American Federation of Information Processing Societies (AFIPS), with broader scope and greater flexibility. As you know, AFIPS is a society of societies organized to represent through a single body the professional societies of the American computer and data processing world. The enthusiastic response to the formation of AFIPS is highly gratifying and lends encouragement, confidence and a sense of mission to those whom you have charged with conducting its activities.

There are times when the path to the future is best appreciated through a re-examination of the past. I would like to quote from a letter dated December 15, 1959, written by the late Chairman of NJCC, Professor Harry Goode, who contributed so much both to NJCC and to the birth of AFIPS:

"I believe the major objective in the formation of the society is to provide for *information flow* in all other instances than those provided for by the individual societies to their members.

"There are four types of such flow:

(1) Information flow between members of information processing societies nationally.
(2) Information flow between our national information processing society and foreign information processing societies.
(3) Information flow between societies in the information processing profession and other professions.
(4) Information flow from the information processing societies to the general and educational public.

"If we can recognize a firm set of objectives such as these (which of course need to be rewritten into a proper set of words), then what the society is to do is relatively clear-cut.

"The functions follow immediately from the objectives:

(1) Act as the American representative body on matters related to computing application and design, in a broad area of computational and information processing sciences.
(2) Advance the field by stimulating research into new aspects of computer sciences emphasizing the cross-pollination of ideas among member societies.
(3) Prepare, publish, and disseminate information of a tutorial nature to laymen, high school teachers and students, government officers and officials, etc.
(4) Maintain relations among American and foreign technical societies through conferences and symposia, cooperation with other societies in organizing sessions at their conferences; provide reference material to other societies on the computational sciences.
(5) Maintain membership in the International Federation of Information Processing Societies (IFIPS).
(6) Aid in certain actions of member societies involving participation and cooperation by more than one society.
(7) Sponsor the JCC's."

The Constitution of AFIPS reflects these views in their entirety. With your frequently demonstrated cooperation and support, the Board of Governors of AFIPS will continue to conduct our successful Joint Computer Conferences and to represent the United States in our International Federation, IFIPS. As new societies join the Federation, it will gradually provide the hoped-for broad representation of the American information processing profession. We will seek to establish AFIPS as the information center on data processing including not only bibliographies of written material, but also a calendar of events of computer activities in the United States and throughout the world, a roster of individuals active in information processing, and a current file of developments in progress or recently consummated. We plan to establish a speakers' bureau to carry information on the information processing field to educational institutions and professional societies. We plan to establish a public information committee which, through the media of personal contacts, press releases and tutorial articles, will make available to laymen, to government agencies, to affiliated and member societies and to the profession as a whole, the present status and the probable future of information processing in the United States.

I trust that with your continued cooperation and support our efforts will meet with a long string of successes.

Respectfully submitted,
Morris Rubinoff, Chairman
National Joint Computer Committee

Figure 3

Yes, AFIPS; in order to simplify U.S. representation in the International Federation of Information Processing after its formation in 1960, the National Joint Computer Committee had transformed itself into AFIPS (see Figure 3). I had been involved as an ACM representative to the NJCC in 1959 and 1960, but was getting ready to move to Europe at the crucial moment. I remember Walt Bauer was the chairman of the last pre-AFIPS Joint, in May 1961, and that the death of Harry Goode, who had planned the changeover, cast a pall over much of the action.

I attended IFIP in New York as a recent immigrant; Generous Electric had reached out to Lausanne and hauled me back to lovely Santa Barbara just a month or so before. I had missed the first Las Vegas meeting while overseas, and hence doubly relished the 1965 FJCC. It was *huge;* Asmus was in his glory. The town was fun, to one who had been away for some years. The papers were fresh; I especially remember a panel on the overseas market (Norm Ream, Jim Miles and others)—I commented from the floor that it would be safe to let the Russians have big CDC machines, but only if they agreed to take the software too! And there was a great look-ahead panel that opened my eyes to the coming LSI revolution: Rex Rice had just gone to Fairchild.

This was the first Joint to publish its proceedings in two volumes; the San Francisco 1968 holds the record to date, with 3.5 inches of paper and bindings. The first nineteen Joints had paper-covered proceedings; bound library-style, they span ten inches. The volumes from 20 on, which were bound by AFIPS, span 53 inches: already, we have the proverbial five-foot shelf of books!

At the end of 1962, the IRE and the AIEE, founding fathers of the Joint concept, had merged into the

IEEE. The Simulation Council had become a smaller member, so AFIPS continued to have three partners, albeit rather disparate: the simulation people, the computer part of the giant IEEE, and all of ACM.

A horrid thing happened in 1967. Having outgrown all other eastern exhibit facilities, the Joints began meeting in Atlantic City—the SJCCs, that is. Five out of the six, through 1972, were held there; the 1969 was shoehorned into Boston, but hotel and exhibit facilities were completely swamped, and we retreated to Jersey. I remember the Boston meeting largely because I flew from the last moments of the ACM Council, to which I had been elected the year before, all the way to Tokyo—and made a major speech a few hours later. I was based at the Bureau of Standards by then, and flew the same year to meetings in Novosibirsk; it was 1970 before my jet lag wore off!

Shortly after I came to NBS, I had had the sad privilege of helping award the Harry Goode medal to Sam Alexander—three times. The award was to be made at FJCC 67, in Anaheim, but we all feared Sam, one of the great figures of all the Joints from the very earliest days, might not be able to make it. So we gave him a blank medal at ACM 67, which fortuitously met only a few blocks from his house in August. Then when the engraving was done, the director of the Bureau and I gave it to him again. And indomitable Sam flew out to California in November, and got it again, in formal ceremony. He died less than a month later.

Another departure, less tragic of course, was Charlie Asmus leaving AFIPS. More than any other person, he had built up the exhibits to the huge, multimillion dollar extravaganzas of the Sixties. He and I had worked together, inside and outside the Joints, since 1954. He still attends; he still organizes meetings all over the world—but the Joints missed him.

The last WJCC to meet in San Francisco was in 1968. There were nearly a hundred unaccommodated potential exhibitors, in addition to the 130 who showed. The hotels were swamped; people flew in from Los Angeles and went back the same night. Fees were up, to $20. There were complaints. Las Vegas was better, while the aerospace euphoria of the Sixties lasted, but in the end, with deflation, Puritanism prevailed: too many bosses thought the attendees, papers or exhibits, to the contrary, were really bucking the tables. Not true, in my opinion—but it was a dour time. No more Vegas!

We tried Houston, and the Astrodome complex held us nicely. But the hotel/motel accommodations were scattered; the personal and intellectual and sales contacts suffered. It was the era of the shuttle bus: the Joints were swollen!

They had eaten up the DPMA annual show and the ACM exhibits. They had completely changed the various electronic trade shows: pulled out the major systems, left components and modems and lemon-squeezers. Now they in turn began to suffer. "Too much," the manufacturers said, "We have overseas shows, where the market is skyrocketing, and where sales are actually closed at the booth. We can't afford two huge Joints every year, all the special shows for bankers and retailers, and Paris and London and Hannover." So they began to pull out. The papers poured in as always; the attendance stayed up; the income side of the AFIPS ledger, though, showed the strain. And by this time the Federation included other, less seasoned, societies, societies that were not so sure computers would keep on growing at the old, crazy pace.

So the JCC Committee, of which I was again a member, asked the big boys—the IBMs, the UNIVACS, the Burroughses—what to do. "Cut back to one national show a year, and hold it only in New York or Chicago or Anaheim," we were told. And, moaning and complaining, we did. Growth resumed.

Most of the big exhibitors came back—you will see them at NCC76. We have tens of thousands of attendees at the exhibits, thousands at the technical sessions. Ten or more sessions run in parallel, fiendishly planned (like prime-time TV) so that the best papers compete.

The booth birds are not quite so sexy; Women's Lib has had a say. The recruiting is not quite so vigorous. The hospitality suites are harder to find. Social issues flourish; we deplore EFTS in the panels, and sell its hardware and software like mad in the exhibit hall. Yes, the Joints are gone: the NCCs have taken over. I've been to most of them, and plan to go to many more. Nowhere in the world, not even in Tokyo or Paris or Hannover, do you smell the excitement, see the vigor of our trade as at these gigantic gatherings of the clan. We invented something good: the shared computer conference. It bridged the disparate interests of our organizations, and of the manufacturers. It let us join together in a common enterprise. And it was fun!

# # # #

I would like to express my thanks to the Stanford University Computer Science library. They have everything, and are most kind besides.

Computer prehistory and history in central Europe

by H. ZEMANEK
International Business Machines Corporation
Vienna, Austria

ABSTRACT

An excursion is made into the history and prehistory of computers with special attention to Central Europe. Three historic periods are examined. The period of programmed automata is rich in the development of programmed clocks and musical instruments beginning in the 14th century and continuing through the 18th century. The period of programmed weaving to punched cards ranges from the Jacquard loom to the Hollerith card. It can be noted that the Jacquard card and the Hollerith card are the same width. Predecessors of Jacquard can be found in Austria as early as 1680 or 1690. The Hollerith card was used in census of 1890 in Austria, a fact not as well known as its use in the United States census of 1890. The story of Otto Schaeffler, the engineer who produced the equipment for the Austrian census, is a fascinating one. The third period is the period of programmed calculation. The computer is the product of two streams of development, calculating devices and programmed calculation, and contributions from Central Europe are prominent in both streams. Among the names which must be included are those of Petzval, Boltzman, Goedel, Morgenstern and Von Neumann. Attention is called to two contributions from the author's laboratories, namely, the 1954 fully transistorized computer, "Mailufterl," and the "Vienna Definition Language."

INTRODUCTION

New ideas are rarely really new ideas. Mostly, they are just another step on a long road from ancient times into the future; often an idea becomes known as a new idea, because at this moment the idea was suddenly supported by a possibility of technical realization or industrial exploitation. We tend to underestimate the skill and wit of our ancestors. Most ideas, furthermore, are only a part in a set of ideas which all together makes a scientific field or a technical application a flourishing subject. One "new" idea, in other words, depends very much on many other flanking ideas.

Nothing, therefore, can be understood without a knowledge of the relevant history, and any object, natural or artifact, carries elements and traces of its history. Understanding the present and judging the future of any science or technology, institution or company is only possible on the basis of knowing the past.

So the modernistic view that history is uninteresting except for historians or that history today proceeds so fast that it is not worth looking into the past, is totally wrong.

The impression of speed or acceleration, by the way, is a function of specialization. You might have heard the bon mot about the difference between the universalist and the specialist which says that the universalist knows nothing about everything while the specialist knows everything about nothing. That sounds symmetric—but only until you consider time. Because then the zero knowledge of the universalist holds forever, while the total knowledge of the specialist lasts for zero time. Rephrased less extremely, one could say that the less specialized the longer the truth lasts and acceleration is proportional to specialization.

Computers require a lot of specialization and abstraction, and we all should compensate for that by carefully cultivating our personal universality and by intentionally rehumanizing our technology and profession. This principle encourages an excursion into the history and even prehistory of computers with special attention to Central Europe.

Thus the paper is organized into three sections of different length:

(1) the period of programmed automata,
(2) the period of punched card development, and
(3) the history of programmed calculation.

PROGRAMMED EARLY AUTOMATA

Time is a steady flow without steps, but in order to indicate or measure it, steadily working or analog devices are less accurate than digital ones. This basic principle is very true of the computer, but it was first discovered for clocks and watches and implemented in the forms of the pendulum and the balance wheel. Once the step function is introduced, it takes a small step to use programs and to add programmed devices. The stroke of the full hour and of the quarter soon is followed by automatically moved figures or by chimes.

Music, of course, is typically digital, quantized in frequency, time and even in amplitude—from pp to ff. The musical notes and the score are digital programs, their realization in automatic musical instruments is well within the spirit of music, and the classic composers like Bach, Haydn, Mozart and Beethoven have written for automatic music machines.

In Northern Germany and in the Netherlands we find the oldest chimes, dating back to the 14th century. In Southern Germany, particularly in Augsburg, automata of very fine artistry have been produced during the 16th and 17th centuries. Museums in Dresden and Vienna are exceptionally rich in automata of this period. We have in Vienna even a programmed automatic piano of around 1600.

The 18th century brought great advances in mechanics, and the art of building automata and music machines flourished in Switzerland, Southern Germany and Austria—but also in France and England. Again, the museums in Dresden and Vienna are exceptionally rich, but one must also see the Swiss treasures in Neuchâtel, Lelocle, La-Chaux-de-Fonds and Auberson. It would be my pleasure to show many of those automata and to explain their history and their mechanisms, but I want to cover a much broader field and I must, therefore, restrict myself to a few hints. A couple of pictures, however, of the most beautiful automation I know, the so-called Knights-Fight-Clock in the Vienna Imperial Castle, I cannot suppress. This mechanical theater was built in the first third of the 18th century. The gate is only four inches high and the performance consists of the presentation of the role-cast including a small brass band on horse followed by three fights, where in the first fight the right knight is pushed from the horse, in the second fight the left knight loses his helmet and in the third fight the lancet of the right knight is broken. The last event is the piercing of the Moor's head, and because this scene is so nice, it is done twice. Before the gate closes again, the spectator gets a look at the rearrangement of the men.

FROM PROGRAMMED WEAVING TO PUNCHED CARDS

Programmed weaving is generally connected to the Jacquard loom, because Jacquard not only refined this technology but also brought it to industrial production and application during the Napoleonic time. But the ideas are much older.

Weaving is of more interest for information processing than we attach to it. That it is binary stems from the fact that each crossing of two threads means a natural digital-point. Many folkloristic weaving devices—in Europe, but also in Africa and Asia—are implementations of or tools for programmed processes. And weaving is in particular important in our present-day advance from serial to parallel processing: weaving, in contrast to mathematics, is a naturally parallel process and might give us more ideas than we think.

It is known that Jacquard had many predecessors, J.B. Falcon and B. Bouchon for instance, and Vaucanson, who is known by his two music players and his artificial duck, but whose main contribution was the preparation of Jacquard's success, the development of a punched tape-controlled loom.

Recently one of my Austrian friends discovered, however, a device which seems to be considerably older. It is called "Bröselmaschine," and there are two programming units in existence, both in the province of Upper Austria. They were made around 1740, and there are good indications that the invention was made between 1680 and 1690. Wooden bars are glued on a closed loop strip of linen, and the bars operate the weaving device. The name, by the way, does not seem to be derived from Brösel (crumbs), but from the diminutive form of Ambrosius, probably the family name of the inventor. It is interesting to note that Jacquard punched cards are of the same width as today's computer punched cards: certain measures change little over long times. There is the nice story that Mr. Watson answered the question of Mr. Hollerith what format he should choose for the punched card by opening his wallet and producing a dollar bill (of that time). But in all probability, Hollerith chose the size of the bill in order to be able to make use of certain sorting devices already in use for dollar bills, or much simpler: he started from a jacquard punched card in his development and found no reason to change the width.

Everybody knows that punched card equipment was introduced by Hermann Hollerith for the US census of 1890. Much fewer people know that the idea very probably comes from John Shaw Billings, a medical doctor of the US army assigned to the census office. He triggered Hollerith to construct census machines after the Jacquard loom idea; you will find this story in Herman Goldstine's book "The Computer from Pascal to von Neumann". Even less people know that there was one country where the census of 1890 was also done on Hollerith machines, and that country was Austria.

Again, it was not an engineer who triggered the enterprise, it was an economist of reputation, Professor Inama-Sternegg, head of the Austrian census office. How he had heard of the American development we can only guess, but Hermann Hollerith was on his honeymoon trip in Vienna at the right time, and it is probable that Inama got the idea by Hollerith's visit in the Vienna census office.

The engineer who made Inama's intention possible, who cooperated with Hollerith, introduced and produced equipment and maintained it, and who finally made a very big step forward, deserves a section of this paper. I dug up this man, so to speak, (who had been practically totally forgotten in Austria) during three years of a fascinating research. Here, I can give only a short version of my findings.

OTTO SCHAEFFLER

(Theodor Heinrich) Otto (Hermann) Schaeffler was not a native Austrian. Like many famous Viennese—Beethoven, Brahms, Maelzel—he came from abroad. He was born on October 15, 1838 to a Wurttemberg pastor family in Unterheimbach, east of Heilbronn, Germany. His parents sent him to the priest seminary of Blaubeuren near Ulm, but the boy wanted to become a mechanic. So he left the seminary at the age of 15 and entered a mechanics shop in Stuttgart. His aptitude for this profession is confirmed immediately. He received an award of second class in the first year and an award of first class in the second year, the latter for designing an electro-motor. In 1855, he traveled to Vienna and continued to learn in mechanics shops for four years. He then went in 1863 to London for another four years. He settled in Vienna for the remainder of his life.

He understood how to grasp opportunity. When the Austrian Post Administration in 1867 bought (for 40,000 Austrian Guilders) the patent rights for the Hughes printing telegraph, which, in the following year, was internationally accepted at the Second International Post Congress in Vienna, Schaeffler cooperated with the American inventor so successfully that he could start local production of Hughes telegraphs. He soon exported them to Serbia and Roumania, to Italy and Switzerland and even to Japan. Schaeffler established his own factory and in an advertisement of 1871 offered all kinds of telegraph equipment, railway signalling systems and physical measurement apparatus. The main customer was the Post Administration which in 1871 closed their own Central Telegraph Workshop (founded by the German inventor and Telegraph officer Steinheil in 1850) and turned to private suppliers. Schaeffler succeeded in getting the main contract, installed a contract workshop in the Post Administration building and ran the business of Post Telegraph and Telephone supply and maintenance until 1896. His successors continued this work until 1913. He supported the projects of the Post engineers. He let them publish; he let them earn all the glamor he produced and he sold. So there is little trace of his work in the technical literature; I had never heard his name during my studies and I had a lot of work to collect the facts about him. The result is fascinating.

At the Vienna World Exhibition of 1873 and at the Paris World Exhibition of 1878, Schaeffler showed, apart from his Morse and Hughes telegraph equipment and his railway signalling systems, a stock exchange printer of his own invention. His exhibit was one of the best at both events; he received gold medals and France made him Chevalier de la Legion d'Honneur and Officier de l'Academie.

In 1874, Schaeffler invented another printing telegraph, a quadruple system like the Baudot, but mechanically more sophisticated. The Hughes telegraph had two synchronously rotating fingers, one in the sender and one in the receiver. By a piano-like keyboard the operator selected a letter and thereby made contact with the rotating finger in the corresponding direction. Since the receiving finger was in the same direction at this moment, the receiver could print the correct letter. The Baudot and the Schaeffler printing telegraphs use a five-bit binary code. But while the Baudot operator must learn the code and apply it to a five-key board, Schaeffler had a Hughes-like piano keyboard for 26 letters (or signs and numerics) plus letter blank and sign blank; below the keyboard there are gliding bars like in a tele-typewriter of our days which produce the code. Schaeffler's code is a reflected binary code! What F. Gray patented in 1953 for PCM, Schaeffler had applied in his telegraph in 1874, and for a similar reason: reliability. He had contact fingers sensing on five cams consecutively all combinations; the right one triggers printing. If the fingers are to make a minimal number of movements, the solution is the reflected binary code. For Schaeffler, this idea was a minor one. More exactly, the code is described in a letter by the Austrian Post employee, J. N. Teufelhart, inserted there as a footnote and telling that Schaeffler found the code by combining wooden bars with the different combinations until he had the best solution. Another Post employee, Alexander Wilhelm Lambert of Linz, claims to have shown this code to Schaeffler as early as 1872, but this claim is not clear and cannot be checked.

The Baudot apparatus was successful, the Schaeffler apparatus was not; the Post Administration was not interested in proliferation of telegraph systems. Schaeffler soon turned to the next subject, where he saw and obtained his next big chance: the telephone.

Graham Bell and Elisha Gray had shown their inventions in 1876 at the Philadelphia World Exhibition. Two years later the first American telephone network opened in Detroit. Vienna followed in 1881. A private company got a license for a telephone network within a 10-mile circle around St. Stephen's Cathedral and started in December 1881 with 154 subscribers. Schaeffler constructed and built telephone stations. He supplied the exchanges for 500 subscribers in 1882, for 2,400 in 1884, to which he adds a second of the same capacity in 1890 and a third for 3,000 in 1892. In a report by J. Hopkinson, a member of the Royal Society of London, dated 1893, the Vienna network is classified as faster, better (except for some noise in the lines), and less expensive than the networks of London, Paris and Berlin. In 1895, the Post Administration bought the Vienna network (which now has 18,500 subscribers) for 1,300,000 Austrian Guilders, so that the Austrian telephone system was completely nationalized.

In 1880, Schaeffler married a Viennese and in 1883 became an Austrian citizen. Between 1874 and 1895, he filed 18 patents and in 1885 he fought the Bell patent with partial success. In 1884, he moved into a new factory where he employed 80 workers (plus 6 in the contract workshop at the Post Administration). He was known in Vienna as a progressive entrepreneur. He allowed no children's work and no Sunday work, which was not a normal procedure in those days. He financed health and employment insurance for his employees and was interested when they got medals (from the trades' union of lower Austria) for working 25 years in their profession.

In 1889, Hermann Hollerith received his patent for the punched card system. The director of the Austrian Census Office (Central Statistical Office), Professor Inama-Sternegg, an important statistical and economic scientist of his time, heard about the new machinery and wanted to apply it to the Austrian census of 1890. Schaeffler accepted the task of importing and servicing the Hollerith machines, providing for the power supply, organizing trial runs and using it for smaller tasks like cattle census and hospital statistics. In the spring of 1891, Emperor Franz Joseph I visited the Hollerith operation which processed 28 million punched cards using 12 machines during 667 days, thus carrying out almost 100 million counting steps. The emperor was very satisfied with his visit and after completion of the work, Schaeffler received a very high distinction (Ritterkreuz des Franz-Josephsordens).

Programming of punched card operations was extremely clumsy on Hollerith's early machines. An electrician had to wire the interconnections between sensors, counters, the relays and their contacts. Schaeffler, the experienced telephone exchange specialist, saw the way for the remedy and applied exchange technology, such as plugs and plug-in cables. On May 20, 1895, he got the Austrian patent No. 463 182 for this idea. He wired the different elements around the bar to make contact on plates of metal. He used 77 counters, 100 relays, 240 punched card hole sensors, and 5 batteries which were accessible and could be programmed by connecting neighboring elements with plug in cables and groups of sensors with metal sheet forms. The programming board could be moved out and the electrician had access in case of trouble. This patent clearly was the beginning of technical programming which could be carried out by the census employees without a requirement for an electrician.

In 1896, Schaeffler came home with a big surprise for his wife. He had sold his factory to Czeija & Nissl who then used the name United Telegraph and Telephone Factories Czeija, Nissl and Company and is now merged into ITT Austria. Schaeffler had requested the term "United" in the name to indicate the succession and, in fact, the new company continued the contract with the Post Administration until 1913. Schaeffler's big 7,800 subscriber exchange was replaced in 1898, but in remodeled form it was installed again in Prague and worked there for several years.

Before retiring, Schaeffler moved into an imposing house behind the factory he had built. Even today the guest of the present owner, who organizes chamber music concerts in Schaeffler's ancient rooms, are excited about the place.

Schaeffler died in 1928, at the age of 90, unknown, not as rich as he was at the end of the century, but after a life full of success and satisfaction.

With Schaeffler's retirement, the import and production of punched-card equipment obviously was at a provisional end; the machines of the Census office continued to work, the population register was made a continuous service operated on Schaeffler's punchers, counters and sorters. One or the other machine was still in existence at the end of the Second World War.

GUSTAV TAUSCHEK

A generation later, development started again. Gustav Tauschek (1899-1945) began around 1930 to develop accounting machines of a new type on punched-card basis. He used only the upper half of the standard card for a punched one-out-of-ten code, while the lower half served as written or printed document. There was a sorter running at 20,000 cards per hour; the calculating machine had 75 places for counting and printing, all four basic operations could be performed and also the total sum of the digits of a number. The throughput was at 4,000 cards per hour. Programming was done by plugboards.

Tauschek's machines remained single models, no production was made. But Tauschek sold 169 of his 200 patents to IBM, among which there was a set for an interesting reading machine.

JOHANN NEPOMUK MAELZEL

Another Austrian (again not born there) I want to mention shortly is Johann Nepomuk Maelzel whose life story has not yet been written. I have collected almost all material and I hope to write it soon. We cannot cover here his contributions to the Music Machine of the 19th century and his success in Artificial Intelligence. Maelzel had invented the Panharmonium, the first flute organ extended by trumpets and percussions, and the first automatic trumpet player. Maelzel had pushed Beethoven to write a Battle Symphony, the first stereophonic composition for two automata, intended for Maelzel's machines, but never performed on any. Maelzel himself convinced Beethoven to rewrite "Wellington's Victory" for two orchestras. The cooperation ended in a law suit which, however, was terminated by a friendly agreement. Maelzel bought the famous wrong Chess Player from the son of Herr von Kempelen, sold it to Eugen Beauharnais and got it back in 1817. Nine years later, he came over here to the US and performed with the Chess Player and other automata in simulating artificial intelligence. Kempelen's trick, to prove that an obviously man-driven chess player could not possibly contain a human being, was superbly staged by Maelzel. Once he was dead, the chess automaton was practically dead. That it was destroyed by fire in Philadelphia in 1854 is only the proper dramatic end of a fascinating story.

Maelzel, however, also introduced the master clock into music, and a standard for musical speed, much more efficiently than any modern standards committee. He started off by stealing the mechanical invention from D. N. Winkel, who had invented his chronometer in 1814. Maelzel not only got patents in London, Paris and Vienna, he started fabrication in all three places within two years after stealing the mechanical construction idea. Apart from his sales campaign, his main contribution was first, a good trade name, metronome, and second, the scale which made the device attractive. From the natural marks at 60 and 120 beats per minute, he derived a 16-step scale which is accepted by musicians to this day. All of that was accomplished in two years. Maelzel would give metronomes as presents to the most famous composers in London, Paris and Vienna, if they signed a declaration that they would mark future compositions by the Maelzel measure.

DIEDERICH NIKOLAUS WINKEL

D. N. Winkel (1777-1826) was very angry about Maelzel's total disregard of him as the real inventor. (It is true, however, that Maelzel made the mechanical idea a world success, which Winkel could never have achieved. This is a lesson for the engineer on the difference between a technical idea and its commercial exploitation). So Winkel decided to beat Maelzel in his (then) central field: in the field of musical machines. And Winkel built a music machine which could compose, and this is the first construction I know of to use a stochastical element. The trick is to distribute eight forms (the melody and seven variations) of a composition over two pin drums, two bars on one and two bars on the other in sequence. The stochastic element decides a stochastic path through the stored information which would repeat itself only after 5 million years, as the Paris Academy calculated.

The stochastic element is a 3″ wheel where the first and third quarter of the circumference is cut out. It is started and slowed down. When it stops, a finger finds out the decision (stopped at a cut-out quarter or at a full quarter) and correspondingly commands a move or not into the adjacent variation. Weaving, Music Machines and Census have prepared as many ideas for the computer as calculating devices. Only part of the history of ideas is collected and described in publications of our days and in our language. A lot remains to be done.

But let us now turn to programmed calculation.

PROGRAMMED CALCULATION

The computer, in fact, is the product of two streams of development, namely calculating devices and programmed calculation, and it should never be forgotten that a computer can be as well man as machine.

The prehistory of calculating devices is well known, from the calculi, the reckoning-stones to the abacus, which might be an Asiatic invention but might also come from Europe. The contributions of Pascal and Leibniz are known, but the fact that a professor of theology in Tuebingen, Germany, by the name of Schickart built a four-species device already in the year in which Pascal was born, had to be rediscovered several times. No Schickart machine survived, but we have some documents from which several reconstructions have been derived. The device contains an adder with carry plus cylindric multiplication tables which permit easy adding for multiplication and substraction for division. Since transmission is done by the operator, the system is less reliable than the later desk calculators.

Leibniz, by the way, not only described the binary system, which he got from the Chinese philosophical code connected to I-Ging, Leibniz also described the construction of a binary adder operating with metallic spheres. The case 1+1 yields zero, carry one, requires channelling one of the two spheres to the next binary place, while the redundant sphere falls through into the container which from time to time must be "poured" by hand into the "stock" above the added. Dr. Mackensen of the German Museum in Munich has built a model, restricting himself to the technical possibilities of the time of Leibniz.

But let me turn to programmed calculation. Certainly, there were many examples for it—let me tell you an Austrian story which had very practical consequences. Around 1840, there was a typical Viennese University Professor by the name of Joseph Petzval who was born in a German town in a Slovak province of the Kingdom of Hungary. He had worked in several fields of Applied Mathematics and Physics, and he had written an interesting mathematical theory of music.

At that time he had an idea how to produce high-quality lenses which would make the newly invented photography a real art and technology. But his idea would have required so many hours of calculations that a full crowd of calculators would be required which a University Professor never could afford to pay.

It was the Army who helped. One day, Petzval told his story to a Hapsburg, Archduke Ludwig, who then was the director of the Austrian artillery. Using bombardiers to calculate firing tables was already standard.

Some of these were very experienced specialists in numerical calculations. Petzval borrowed some of them and the result was first a landscape lens and later a portrait lens, manufactured by a German optician Voigtlaender who obtained a world-wide reputation based on these programmed calculations.

Petzval, to my knowledge, was the first one to use the term "thinking machine" and he used it properly. In his introduction to the paper on the programmed calculation of the lens, he says that pure mathematics is the construction of a powerful thinking machine.

A similarly powerful thinking machine is logical algebra, reinvented by engineers under the name of switching algebra. From 1895 to 1936, there is a full series of Austrian forerunners of Shannon. A contribution to the thinking tools around computers comes from Ludwig Boltzmann, the Austrian physicist who lectured on thermodynamics. His chapter on entropy contains a derivation of Shannon's formula $H=p_i \log 1/p_i$. (It is from Boltzmann that Shannon took over the letter H for entropy.)

Austria's contribution to the theory of computation is the revolutionary paper of 1931 by Kurt Goedel, then Assistant Professor at the Vienna University, on the undecidability of axiomatic systems above a certain degree of complication. The theory of games and economic behavior comes from Budapest-born John von Neumann and Vienna-born Oskar Morgenstern. The philosophy of information processing can be based, as I have shown, on Ludwig Wittgenstein's "Tractatus Logico-Philosophicus," written while he was in the Austrian Army. All these ideas are as important for the development of information processing as the actual construction of computer models and computers.

Let me close this paper with two paragraphs on Austrian contributions to the electronic computer which were achieved in my own environment.

In 1954, we started at the University of Technology in Vienna the development of a fully transistorized computer which was given the name "Mailüfterl" after a friendly Viennese May breeze—thus indicating its more modest parameters compared to the ambitious computers of those days called Whirlwind, Typhoon and Hurricane. But this Viennese development (which finally was, e.g., not that slow; with hearing aid transistors we could run it at a clock frequency of 130 Kilocycles per second) had several interesting features. The instruction word included van der Poel's functional bits (nine of them, in fact), could decide between binary and decimal operation, use flag-bits and attach one of 15 different conditions to any instruction.

The second development was the formal definition of syntax and semantics of PL/I, including the design of the Vienna Definition Language and an Abstract PL/I machine. The ANSI Standard for PL/I is written in this Vienna Definition Language, although there have been critical objections to this abstract method. But there is no easy way to describe a language of the size and sophistication of PL/I in English or in a half-formalized language. I think it is impossible, as a matter of fact, if one is unwilling to accept serious ambiguities. Obligation to formalization is as old as formal methods; reluctance is as human as the invention of formalisms. All is a matter of education, balance and respect for the human aspects of science and technology.

I have presented a sequence of highlights rather than a systematic paper. My excuse is that the material that exists on the subject is enough for many hours of lecturing and would require years of work to render it in a systematic form. It was not my intention to bore the reader with details. I wanted to invite him to glimpse fascinating adventure, to see not only more than the circuitry and the present status of the computer, but to understand this magnificent invention, the computer, as the result and combination of many hundred years of human efforts of many kinds. To trace these efforts back in history is as fascinating as it is useful.

Early computers in Europe

by RICHARD WILLIAMS
Computer Consultants (International) Limited
Llandudno, Gwynedd, Wales

ABSTRACT

This paper describes the early history of computers in Europe, notably in Germany, Holland, France, Italy and the Scandinavian countries as well as Great Britain.

Of necessity, in such a short paper, information is given in a fairly short form, but the paper also includes a detailed description of the birth and foundation of the most successful first British commercial computer company—Leo Computers Limited, and this gives an insight into the thinking which lay behind British early computer development.

Three appendices are included which give the names and addresses of the early computer manufacturers and sales organizations in Europe, and short notes on the early computers and calculators.

The development of electronic computers started in Europe earlier than it did in the United States of America.

Developments in Great Britain occurred in the late 1940's at the end of the Second World War, but prior to that Konrad Zuse in Germany, as far back as 1934, started development work on program-controlled machines, and in 1937, jointly with Dr. Schreyer, started development work on electronic computers proper. By 1941 they had completed the first fully operating Model Z3 in electro-mechanical technique; program on punched tape, binary system and floating point arithmetic.

During the Second World War, Konrad Zuse was involved in developing special devices and an improved universal computer, the Z4, with a mechanical memory. These devices were, in fact, the control mechanisms for the V1s and V2s which almost played a decisive part in the war. Fortunately for the British, Zuse had as many bureaucratic troubles as they had encountered, and the delays seriously affected his work.

The firm of Zuse KG was formed in 1949 and the development of other computers, the Z9, Z11, Z22, Z23 and Z25 took place.

As a matter of record, in 1964 Zuse KG became part of the Siemens empire and by 1969 had become wholly owned by Siemens.

It is strange, however, that apart from Zuse's efforts, comparatively little effective computer development took place in Germany on a commercially viable scale, although by now a great deal of IBM equipment is built in Germany.

What was happening elsewhere?

In Italy, Olivetti were becoming involved, in the early 1950's with the development of machines which were variations on their established commercial equipment, but their development was restricted by virtue of the fact that these machines made use of punched paper tape of Olivetti's own design, which had, in fact, square and not round holes and which had no clock track. This made the equipment as a whole incompatible with the competitors' equipment and this was markedly to the disadvantage of Olivetti.

In France, a considerable number of small electronic companies were trying to get on to the computer band wagon. The dominant company was Compagnie des Machines Bull, which could not quite make up its mind whether it wanted to outdate its existing punched card equipment by introducing computers, or whether it should disregard the future of computers altogether.

Everywhere there was confusion between just what a computer was and what a calculator was. The confusion was added to in Europe, as in America, by different tax structures, depending on whether a machine was, in fact, a computer or was classed as a calculator, and the universal practice grew of giving computers names made up from initial letters. The letter 'C' in a name could stand for computer or calculator as it suited the mood and purpose of the time.

In Holland, the Philips organization was involved in prototype computer development and a completely separate company later acquired by Philips and Electrologica, produced a small number of quite viable machines. Philips' problem was one of administration and bureaucracy, coupled with the fact that they supplied from their numerous other companies component parts which were used in computers and the sale of which they had no wish to lose on the basis of a bird in the hand is worth two in the bush.

In Denmark, three types of computers were built at a very early stage. These were reasonably successful, and would have been even more successful but for Denmark's inability to export them in significant numbers.

The Danish Institute of Computing Machinery built the DASK computer and the GIER computer. The GIER was the first built by Disa Elektronik in Herlev and was known as DISADEC. It was later taken over by the Danish Institute of Computing Machinery in Copenhagen and re-christened GIER.

Gallo Electronics also built a computer called GALLO which, although developed at a very early date, had, in fact, an early demise.

Computer development also took place in Norway, as well as Sweden, and this was geared to the activities of the Great Northern Telegraph Company and its equipment.

One of the fundamental problems with the computers mentioned in the countries above was that the countries all had their own national language and, because of the cost and lack of foresight of the companies concerned in realizing that to sell their machines outside their own country, they had to have a universal language, which as it turned out in the computer field is English, the difficulties of production and economic viability were very considerable.

The countries which had no such language problem were the United States of America and Great Britain, both of which produced computers, sold and operated them, making use of the international language of English, although American English, it must be said, is a little different from British English, both in its spelling and connotations.

Computer development in Great Britain started fairly soon after the Second World War and was particularly based on Manchester University and Cambridge University with off-shoots taking place in the Midlands and London.

It became very fashionable, and still is so, to say that Charles Babbage who about a hundred years ago was a mathematics professor at Cambridge was the father of British computers. This is one of those jolly myths which it is nice to have but is somewhat different from the truth. Presper Eckert told the author that neither he nor John Mauchly had heard of Babbage's Engines at the time they first started work on computers, and probably the same comment might have been made by those people who were working on computers at Manchester, although their Cambridge colleagues could, of course, be expected to have had some influence from Babbage.

Babbage was an individual who was very much misunderstood. He had relatively limited funds and he was accused by his critics of never finishing anything. What actually happened was that before he had finished building one of his many machines, he discovered a better way of doing things and, therefore, there was no purpose in completing that particular machine just for the sake of keeping things orderly. He had a girl-friend who was probably a better mathematician and engineer than he was, and it could well be that such reputation as Babbage enjoyed was as a result of her efforts rather than his own.

The original emphasis in Great Britain, as indeed it was elsewhere, was on the design and production of computers for scientific purposes, and those people involved in this activity delighted in using mathematical terminology instead of simple English, to confuse the public at large and enhance their own reputations and prestige. This had a back-lash effect in Great Britain in that the commercial use of computers was delayed for several years because businessmen did not understand them or realize their potential. Indeed, it was in Llandudno where the author lives, at a conference during the 1950's, that Professor Hartree told the assembled audience that, in his opinion, computers would never be used for business purposes because there would not be enough scientists to operate them. Fortunately, this state of affairs did not exist to the same extent in America, largely because of the practical ability and outlook of Eckert and Mauchly and the fact that Eckert, in particular, had a degree in business studies as well as in electrical engineering, and better understood the thinking and outlook of the American businessman.

Fortunately, in Great Britain also, we had some people of influence who saw a commercial future for computers. In particular, the name of Vivian Bowden, now Lord Bowden, and Principal of the Manchester College of Technology springs to mind. The author was privileged to enjoy his confidence when they published books at the same time, the books being very different. Bowden's was called "Faster than Thought" and the author's simply "The Electronic Office" which was a description of the use and possibilities of computers.

Since 1957, when Computer Consultants Limited was formed as the first independent consultancy company on computers in Europe, that company and its associated companies have produced almost a hundred internationally recognized reference books on computers which, because of the nature of the industry and its continuous change, had, of necessity, a very short shelf life. It is from these books that the appendices which form part of this paper have been taken, with the permission of the companies concerned.

The scientific computers developed at Manchester included MADAM and at Cambridge, EDSAC, and the English Electric Company produced an early computer called DUCE which had its origin largely in Cambridge.

Manchester University co-operated with the Ferranti organization and produced early machines like the PEGASUS, while EMI relied to a great extent on their own research and produced EMIDEC.

Elliott Brothers at Borehamwood showed a lot of initiative in producing scientific computers and at a very early stage sought to find ways of using these for commercial purposes with some success, but without the commercial sales back-up to further the enterprise sufficiently in a hardening market.

There were two punched card companies in Great Britain—the Power Samas organization and the Hollerith organization. Both of these companies invested large sums of money in developing machines which they were not quite sure whether to call computers or calculators, and both were surprised to find that what they had produced was already out of date. It is said that it was at that time that Prince Phillip made the pertinent comment that "if it works, it's obsolete." To seek to protect their position, the companies merged to form ICT—International Computers & Tabulators Limited, and some of their early machines made an impact, albeit a short one because of their inability to keep up with research and development.

It may seem strange to an American audience, but IBM had little or no impact in Britain at that time and, indeed, their British organization was very small. This, of course, has now changed, and by now other significant American companies such as Honeywell have made their appearance.

The author was privileged to be asked to make suggestions about Honeywell's entry into Britain, and later Europe, and amongst his working papers are reports that were prepared both for this development and for other significant mergers of European computer companies.

Unquestionably, the greatest single impact made on the British computer field was by an organization that was not concerned with computers at all. It sold cakes and bakery commodities, had some two hundred shops in the London area, and found difficulty in deciding what to bake each night. It was by virtue of the fact that they could not find a suitable computer to carry out their work that they came to design and build their own, and formed Leo Computers Limited. They successfully built and used Leo I for many years, and later produced very advanced versions of the machine before the company ultimately merged with other British computer companies and came under the control of ICL, the, by now, sole remaining British computer company.

The National Cash Register Company had also developed its own computers which were marketed in Britain, and an arrangement was entered into between that company and Elliotts to assemble NCR computers at Elliotts' factories in Britain. This arrangement was expanded to allow for Elliotts' continuing to sell machines for scientific purposes with the same machines, notably the Elliott 803, being sold for commercial purposes by NCR. This arrangement cannot be said to have been altogether successful.

However, to return to the Lyons story. In 1947, two men called Standingford and Thompson, both employed by the Lyons Organisation, decided to study the possibilities of electronic calculators in the office and, to this end, arranged to visit Dr. Goldstine of Princeton University.

Professor Hartree, who was also working on computers at Cambridge heard of this visit and invited Standingford and Thompson to visit him and Dr. Wilkes at Cambridge. Following the visits to Cambridge and America, Thompson and Standingford prepared a report for the Lyons Board and proposed that the company should take an active part in promoting the commercial development of electronic calculators. The Lyons Board decided to donate £2,500 to Cambridge and to lend a man for six months to assist Dr. Wilkes in the activity. Lyons continued their studies during 1948 and the building of EDSAC, the first electronic computer to be built at Cambridge, was nearing its completion.

By November 1948, Lyons had prepared an experimental payroll program which was to be tried out on EDSAC as soon as the computer was ready.

It became quite clear to Lyons by now that it was necessary for them to hold the initiative in this activity and, to do so, they needed staff of their own who were capable of acting on behalf of Lyons. To this end they advertised for the services of an electronic engineer and late in December, J. N. M. Pinkerton was appointed.

By January 1949 they had made practical arrangements for installing a calculator, as it was then called. The clerical staff were told that it was intended to start a computer project, and during the whole of the activity Lyons maintained an excellent staff relationship by keeping their senior, middle management and junior staff fully in the picture as regards their intentions and the progress the activity was making.

For a commercial computer, it had now become obvious to Lyons that it was necessary to have more input/output devices than were customarily intended to be involved with these scientific machines. In April, 1949, therefore, they recommended and, in May, implemented arrangements with Standard Telephones and Cables Limited at Enfield to develop, on their behalf, certain input/output devices.

In May, 1949, EDSAC did its first job of work and a significant step forward had been taken. By July, 1949, Lyons had entered into an arrangement with Wayne Kerr Laboratories Limited to produce panels of electronic circuits on their behalf and to their specification. By August of the same year, Standard Telephones had carried out a survey to ascertain what work was necessary to produce the input/output equipment, and a Mr. E. J. Kaye joined Lyons as assistant to Pinkerton. By the end of the month it had become apparent to Lyons that considerable research was required into the way the clerical work should be organized for the calculator and into techniques of programming. As a result, a start was made on a systematic analysis of clerical work and also on the basic techniques of data processing and programming. The initiative was very much in the hands of T. R. Thompson and it was his drive and perspicacity which largely led to the success of the operation, but it must be emphasized also that he had had the foresight to appoint excellent people to help him in the activity.

In September, 1949, Mr Simmons, one of the Directors of Lyons, christened the calculating machine "Leo" which stood for Lyons Electronic Office. The Coventry Gauge and Tool Company Limited were asked to commence the manufacture of large delay battery tubes which were going to be the computer's memory.

By the end of the year, Standard Telephones had completed their research work and had indicated the equipment which they considered necessary. All this activity was geared to the production of a pilot Leo computer and, in January 1950, Standard Telephones and the other people concerned with assembling the various units were given their head and the equipment required was to be completed and delivered to Leo by January 1951, that is twelve months later.

During March of 1950, demonstrations were given to the Director, office managers, supervisors and members of the Clerical Staff Committee on the way counting and addition was to be carried out on the calculator, and talks were given to the Lyons office staff on the philosophy of electronics in offices.

The first of the delay tubes were received from the Coventry Gauge and Tool Company and Lyons considered the possibility of having the initial records of their data prepared in binary form to be read by photoelectric readers developed by them for that purpose. The activity was now assuming considerable proportions and in April 1950, Thompson was released by the Lyons Board from his other commitments and allowed to devote his entire time to the Leo project. His first step following this was to appoint Mr. D. T. Caminer to take charge of the programming for Leo.

It is significant that the majority of those who played a part in the early days of Leo, went on to remain with the company, and the tradition of almost joining the Lyons organization for life continued strongly in the computer division also. This differs from the approaches in some other companies. To stay with one company for a very long time may have its virtues, but it also has its drawbacks, because it tends to narrow the experience of the group as a whole unless fresh blood is imported from time to time.

By August 1950, discussions had taken place with Standard Telephones to ascertain whether they should develop Leo on a joint basis, but Standard Telephones were not particularly enthusiastic about this and, in fact, the electronic companies at this time must have been quite diffident about the possibilities of using computers in business, in the face of opposition by such well-known organizations as the National Cash Register Company Limited and other office machine organizations.

There were signs at this time that the question of rights to patents were beginning to raise their heads. It was agreed by Standard Telephones and Lyons that neither party should enter into arrangements for supplying equipment similar to that which had been developed without the agreement of both parties. By October 1950, the progress being made was not satisfactory and was behind schedule. To try and correct this, Pinkerton spent a lot of time at Enfield trying to accelerate production, but this was to no avail. By the end of the year, rethinking was taking place in connection with the Standard Telephones/Lyons arrangement, and Lyons agreed to pay Standard Telephones some additional costs in connection with equipment which had been overlooked in the original specification. Standard Telephones agreed to allow Lyons to obtain equipment previously commissioned from them, elsewhere if it was necessary for them to do so.

While the computer was by no means complete, there were working portions of it which could be seen by January 1951 and, in February 1951, Her Royal Highness, Princess Elizabeth, now Queen Elizabeth II, went to Cadby Hall to see Leo carrying out a simple test program.

By April, it was possible to give demonstrations to the Lyons Directors of Leo doing clerical work, and although they were very behind schedule with their activities, Standard Telephones felt that it would be in the interests of both parties if the patent applications of the two companies were considered together.

In May 1951, a memorandum setting out the scope of the Lyons patents was sent to Standard Telephones and a demonstration was given in June by Standard Telephones at Enfield to some of the Lyons Directors and Executives of the partly assembled auxiliary equipment in operation.

However, despite the difficulties which were arising, by the end of August, Leo was doing some real clerical work, albeit very slowly, and the Cadby Hall Bakeries job was being done on the calculator by September and producing accurate results. However, the difficulties arising from the late delivery of the Standard Telephones equipment had not been resolved by October and, regretfully, Thompson and others discussed the possibility of developing other input/output systems using paper tape in order to keep the work going on schedule.

The Leo computer had engendered so much outside interest by November that the Ministry of Supply was seeking to find out whether Lyons could carry out some ballistic computations on the computer. In November 1951 also, the Cadby Hall Bakeries job, which was the scheduling of an overnight bakery programme for the whole of the production at Cadby Hall, was carried out on the computer completely successfully, and the job was carried out regularly from then on.

At this time, the Leo machine was still being referred to as a calculator and, in February 1952, a wages demonstration program was carried out on the machine using magnetic tape for the first time. The records on magnetic tape were subsequently printed by teleprinter and more outside enquiries were being made for time on the machine, in particular, the Meteorology Department and the Ministry of Supply.

In May 1952, Professor Hartree made the remark that computers would never be used for business purposes with any degree of success. This undoubtedly carried a lot of weight with people in authority in Britain and must surely have delayed the use of computers in Britain for a significant period.

Up to about this time the Lyons machine had used punched paper tape and magnetic tape, but in June 1952 began the collaboration between Lyons and the Hollerith company. Originally Lyons had intended that the computer should be used for their own activities and they were pursuing the requirements for their own work. Their next objective was to carry out the provisioning of all the London Lyons Tea Shops, as they were called, some two hundred and fifty in all. In this comparatively short space of time, they had also developed their own photo-electric tape reader and this was in use. An increasing number of enquiries were being received from outside organizations for the use of the machine and the range of enquiries were substantially wide.

By October 1952 Lyons had decided to go ahead with building their own high speed input/output equipment as an alternative to the Standard Telephones equipment because there were increasing fears that this would prove to be unreliable. By the end of the year, early production payroll work was being done and pay slips, produced on pre-printed forms, were being used within the Lyons organization. In January 1953 came a successful full-scale trial of a payroll program and, although there were minor computer faults which were later amended, the operation was initially a success.

The writer believes that much of this success by Lyons with computers arose from their readiness to face realities, and the fact that they themselves had for a long time, as a company, been engaged in organization and method work, and were aware of the necessity for study and planning their operations, rather than just patching them up and doing them in the same old way, albeit with new equipment. In May 1953, the first tabulator was delivered from BTM and a card feed was also delivered the same month. Both pieces of equipment operated successfully, and later that month a demonstration payroll was carried out, printing the result, not on a teleprinter as previously, but on a Hollerith tabulator. By August a draft plan had been prepared in respect of the London Tea Shops Provisioning Application and sufficient use had been made of the computer to appreciate what improvements could be made to the existing equipment. For example, future mercury delay lines would use shorter tubes in which the pulses would be one quarter of a microsecond in width and so reduce the access time to a quarter of that operated in the early days without any loss of capacity. About this time Standard Telephones decided to abandon the project with Lyons and arrangements were made for the re-siting of the equipment.

By the end of 1953 Leo I was operating more than ninety per cent effectively, and a significant number of people were approaching Lyons from outside the organization with a view to hiring computer time and finding out what their future plans were. These included plans for building Leo II, and in February 1954 the amount of coverage given to the success of Leo I, increased enquiries for the new machine.

Outlined proposals for Leo II were finalized and new equipment such as Ferranti Fast Tape Readers were added to the early machine. Lyons then announced that they were prepared to build Leo II for sale or hire to other organizations and that they were forming a subsidiary company, Leo Computers Limited.

The number of staff on the Leo payroll by August 1954 was slightly under ten thousand, and the work for the provisioning of the Tea Shops had been put on the machine, first of all with forty two shops, and later on with nearly two hundred, and this was a success from the start.

By the end of 1954 Bull tabulating equipment had been added to the computer, and the number of enquiries for the purchase of computers and computer time on Leo's own machines, had grown to significant proportions. By now, of course, Leo Computers Limited, have become part of International Computers Limited through various mergers. In the meantime Standard Telephones & Cables Limited (part of ITT) had, in 1958, built their own small computer, the Stantec Zebra. A new transistorized prototype was built six years later, but the company then withdrew from computers. That Leo Computers Limited was a success story was unquestionable. Why was this, when so many computer companies have run into difficulties? Much must have depended on the quality of the people concerned, but the author believes that it was mainly because they knew from the start what they wanted to do, and they went ahead and did it. And eventually knew when to stop trying to keep up with the big boys.

CONCLUSION

What of the future of computers in Europe? The dominant computer manufacturer is undoubtedly IBM, with Honeywell a close second, and with ICL being moderately successful. All the remaining small, individual companies have been swallowed up by these empires, but as competitors there still remains Burroughs, NCR and other American oriented companies.

The future of any individual computer company is damned by the cost of development and the lack of marketing facilities, but there remains the thought that since the early 1960's, with the advent of computer memories, using laser and other techniques, all existing computers are dated—and the size of IBM, for example, makes it virtually impossible for that company ever to switch from these out of date computers to newer, more imaginative equipment which we know is technically possible, and certainly economically viable.

Who knows what the future is?

APPENDIX I—THE NAMES OF MANUFACTURERS AND SALES ORGANIZATIONS OF COMPUTERS WHICH WERE BEING SOLD IN EUROPE IN 1965 AND SHOWING THE ORIGIN OF SOME OF THE EARLY COMPANIES

Aktiebolaget Addo, Malmo, Sweden.
Svenska Relafabriken ABN., Tyreso, Sweden.
Alwac Computer Division, El-Tronics, Inc., California, U.S.A.
A.E.I. Automation Ltd., Manchester, England.
Bunker Ramo Corp., California, U.S.A.
International Systems Control Ltd., Wembley, England.
Burroughs Corp., Michigan, U.S.A.
Burroughs International S.A., Fribourg, Switzerland.
Cambridge University, Cambridge, England.
Centre National d'Etudes des Telecommunications, Seine, France.
Clary Corp., San Gabriel, California, U.S.A.
Collins Radio Corp., Dallas, Texas, U.S.A.
Compagnie Bull-General Electric, Paris, France.
Compagnie Europeenne d'Automatisme Electronique, Seine, France.
(CIT) Compagnie Industriel des Telecommunications
(CAE) Compagnie d'Automation Electronique
(CSF) Both Subsidiaries of Compagnie de Telegraphie Sans Fil which formed a joint company with Compagnie General Electric.
Computer Engineering Ltd., Hitchin, Herts, England.
Control Data Corp., Minnesota, U.S.A.
Control Data AG., Luzern, Switzerland.
Danish Institute of Computing Machinery, Copenhagen, Denmark.
A/S Regnecentralen, Copenhagen, Denmark.
Disa Electronik A/S, Herlev, Denmark.
Electronic Association Inc., New Jersey, U.S.A.
Electronic Associates Ltd., Burgess Hill, Sussex, England.
Electronic Machine Controls Ltd., Thornton Heath, Surrey, England.
English Electric-Leo-Marconi Computers Ltd., Kidsgrove, Staffs., England.
Elliott Bros. (London) Ltd., Borehamwood, Herts, England.
Facit Electronics AB., Stockholm, Sweden.
Atvidabergs Industrier AB., Stockholm, Sweden.
Ferranti Limited, Hollinwood, England.
Friden Inc., San Leandro, California, U.S.A.
Friden International S.A., Fribourg, Switzerland.
General Electric Corp., Arizona, U.S.A.
International General Electric S.A., Geneva, Switzerland.
General Precision Inc., California, U.S.A.
General Precision Systems Ltd., Ealing, London, England.
General Precision France, Paris, France.
Eurocomp GmbH, Minden, Germany.
Schoppe & Fraeser GmbH, Minden, Germany.
International Business Machines Corp., New York, U.S.A.
IBM World Trade Corp., Paris, France.
International Computers & Tabulators, London, England.
Marconi Instruments Ltd., London, England.
Mercedes Buromaschinen AG., Thuringen, East Germany.
Minneapolis-Honeywell Regulator Co. Inc., Massachusetts, U.S.A.
Honeywell Controls Ltd., London, England.
Monroe International Inc., New Jersey, U.S.A.
Monroe International (UK) Ltd., London, England.
Monroe Calculating Machine Co. France, Paris, France.
Deutsche Monroe/Sweda GmbH, Dusseldorf, Germany.
Monroe Calculating Machine Co. Holland N.V., Amsterdam, Holland.
National Cash Register Co. Inc., Ohio, U.S.A.
National Cash Register Co. Ltd., London, England.
National Physical Laboratory, Middlesex, England.
N.V. Electrologica, Den Haag, Holland.
Olivetti-General Electric SpA, Milan, Italy.
Olympia Werke AG., Wilhelmshaven, Germany.
Philips Gloeilampen Fabrieken N.V., Eindhoven, Holland.
Pisa University, Toscanna, Italy.
Raytheon Corp., California, U.S.A.
Scientific Data Systems Inc., California, U.S.A.
Compagnie Europeenne de Calculateurs Industriels et Scientifiques, Paris, France.
Siemens & Halske Aktiengesellschaft, Munich, Germany.
Societe d'Exploitation et de Recherches Electroniques, Aubergenville, France.
Societe Europeenne pour le Traitement de L'Information, Massy S. et O., France.
Scciete Nouvelle d'Electronique, Paris, France.
Societe d'Electronique et d'Automatisme, Seine, France.
Solartron Electronic Group Ltd., Farnborough, England.
Sperry Rand Corp., New York, U.S.A.
Univac Division of Sperry Rand International Corp., Lausanne, Switzerland.
Standard Elektrik Lorenz AG., Stuttgart, Germany.
Standard Telephones & Cables Ltd., Enfield, England.
Svenska Aeroplan AKT., Linkoping, Sweden.
Technische Hochschule, Munich, Germany.
Telefunken GmbH Konstanz, Germany.
Teleregister Corporation, Connecticut, U.S.A.
Zeisswerke GmbH, Jena, Germany.
Zuse KG., Bad Hersfeld, Germany.

APPENDIX II—LIST OF AND SHORT NOTES ON DIGITAL COMPUTERS USED IN EUROPE IN 1963. (SPECIAL ONE-OFF MACHINES, OF WHICH THERE WERE MANY, ARE NOT INCLUDED)

Name	Manufacturer	Price
Ace	National Physical Laboratory	£ 400,000
Advance II	Advanced Scientific Instruments Inc.	£ 300,000
AEI 1010	Associated Electrical Industries Ltd.	£ 250,000
AEI 959	Associated Electrical Industries Ltd.	
Alwac II	Alwac Computer Div. of El-Tronics Inc.	£ 35,000
Alwac III	Alwac Computer Div. of El-Tronics Inc.	£ 37,000
Alwac IIIE	Alwac Computer Div. of El-Tronics Inc.	£ 40,000
Alwac IV	Alwac Computer Div. of El-Tronics Inc.	£ 50,000
Alwac 800	Alwac Computer Div. of El-Tronics Inc.	

Name	Manufacturer	Price
Amdec	Addressograph Multigraph	£ 30,000
AN/UYK-1	Thompson Ramo Wooldridge Inc.	£ 40,000
Apollo	Ferranti Ltd.	£ 35,000
Arch	Elliott Bros. (London) Ltd.	£ 17,000
Argus 100	Ferranti Ltd.	£ 20,000
Argus 200	Ferranti Ltd.	£ 20,000
ASI 210	Advanced Scientific Instruments Inc.	£ 40,000
ASI 420	Advanced Scientific Instruments Inc.	£ 150,000
AV 41	North American Aviation Inc.	
Atlas		£2 million
Basicpac	Philco	
Bendix D-12	Bendix International Div. of Bendix Corp.	
Bendix G-15	Bendix International Div. of Bendix Corp.	£ 30,000
Bendix G-20	Bendix International Div. of Bendix Corp.	£ 210,000
Bendix G-25	Bendix International Div. of Bendix Corp.	£ 300,000
BESM 1	Made in Russia	
BESM II	Made in Russia	
BISMAC I	Radio Corp. of America	£ 500,000
BISMAC II	Radio Corp. of America	£ 500,000
BRLESC	Ballistic Research Laboratories	
Burroughs 204	Burroughs Corp.	
Burroughs 205	Burroughs Corp.	£ 67,500
Burroughs 220	Burroughs Corp.	£ 160,000
Burroughs 250 VRC and 251 VRC	Burroughs Corp.	£ 140,000
Burroughs 260 and 261		£ 70,000
Burroughs 270 and 271		£ 130,000
Burroughs 280 and 281		£ 140,000
Burroughs B5000		£ 200,000+
Burroughs D825		
Burroughs E101		£ 16,400
Burroughs E102		£ 10,000
Burroughs E103		£ 10,000
Burroughs F2000		
CAB 500	SEA of France	£ 23,675
CAB 600	SEA of France	£ 30,000
CAB 3900	SEA of France	
CE 55	Computer Engineering Ltd.	£ 2,000
CE 102	Computer Engineering Ltd.	£ 10,000
Cellatron SER 2	Mercedes Buromaschinen	£ 9,000
CIFA-3	Institute of Nuclear Physics, Roumania	
CIFA 101	Institute of Nuclear Physics, Roumania	
CITAC 210B	Compagnie Industrielle des Telephones	
Clary DE-60	Clary Corporation	£ 8,000
Clary DE-60M	Clary Corporation	£ 7,000
Computer Control DDP 19	Computer Control	£ 40,000
Computer Control DDP 25	Computer Control	£ 40,000
Control Data 160	Control Data Corp.	£ 30,000
Control Data 160A	Control Data Corp.	£ 50,000+
Control Data 924	Control Data Corp.	£ 110,000
Control Data 1604	Control Data Corp.	£ 380,000
Control Data 3600	Control Data Corp.	£1 million
Control Data 6600	Control Data Corp.	£1,600,000
CXPQ	Philco International Corp.	
D21	Svenska Aeroplan Aktiebolaget	£ 60,000
Datamatic 1000	Honeywell Corp.	£ 584,000
Datatron	Electro Data Corp.	£ 39,800
DB 10	Standard Elektrik Lorenz AG	
DB 40	Standard Elektrik Lorenz AG	£ 50,000
DB 70	Standard Elektrik Lorenz AG	
Deuce I	English Electric Co. Ltd.	£ 45,000
Deuce II	English Electric Co. Ltd.	£ 50,000
Deuce IIA	English Electric Co. Ltd.	£ 55,000
Disadec	Disa Elektronik A/S	£ 41,500
ELEA 2001	Olivetti SpA	
ELEA 6001	Olivetti SpA	£ 29,500
ELEA 9002	Olivetti SpA	
ELEA 9003	Olivetti SpA	£ 236,000
Elecom 120	Underwood Machine Co. Inc.	
Elecom 125	Underwood Machine Co. Inc.	£ 100,000
Electronic Anoc 231R		
Elliott 401	Elliott Brothers (London) Ltd.	£ 15,000
Elliott 402	Elliott Brothers (London) Ltd.	£ 22,000
Elliott 402E	Elliott Brothers (London) Ltd.	£ 25,000
Elliott 402F	Elliott Brothers (London) Ltd.	£ 25,000
Elliott 403	Elliott Brothers (London) Ltd.	£ 100,000
Elliott 502	Elliott Brothers (London) Ltd.	£ 135,000
Elliott 503	Elliott Brothers (London) Ltd.	£ 80,000
Elliott 802	Elliott Brothers (London) Ltd.	£ 17,000
Elliott 803	Elliott Brothers (London) Ltd.	£ 35,000
Elliott 900	Elliott Brothers (London) Ltd.	
Emidec 1100	EMI Electronics Ltd.	£ 180,000
Emidec 1101	EMI Electronics Ltd.	£ 185,000
Emidec 2400	EMI Electronics Ltd.	£ 200,000+
EMI Special	EMI Electronics Ltd.	£ 100,000
EPOS	State Statistical Department, Czechoslovakia	
EPSCO 275		£ 35,000
ER 56	Standard Elektrik Lorenz AG	£ 50,000
ES 92	Standard Elektrik Lorenz AG	
Facit EDP	Facit Electronics AB	£ 120,000
Friden 6010	Friden Inc.	£ 7,850
FX-1		
Gallo		
Gamma 3 & MDE	La Compagnie des Machines Bull	£ 85,000
Gamma 10	De La Rue Bull Machines Ltd.	£ 30,000
Gamma 30	De La Rue Bull Machines Ltd.	£ 100,000
Gamma 60	Compagnie des Machines Bull	£ 500,000+
Gamma 150	Compagnie des Machines Bull	£ 50,000
Gamma 300	Compagnie des Machines Bull	£ 22,000
GE 210	General Electric Co. Inc.	£ 270,000
GE 225	General Electric Co. Inc.	£ 84,000
General Mills EC5	General Mills	
General Mills EC6	General Mills	
George	Argonne National Laboratories	
Hipac 101	Hitachi Ltd.	£ 25,000
Honeywell 290	Honeywell Controls Ltd.	£ 60,000
Honeywell 400	Honeywell Controls Ltd.	£ 120,000
Honeywell 800	Honeywell Controls Ltd.	£ 395,000
Honeywell 1800	Honeywell Controls Ltd.	£ 500,000+

Name	Manufacturer	Price
IBM 305	IBM Corporation	£ 65,000
IBM 305 II	IBM Corporation	£ 40,000
IBM 630X	IBM Corporation	
IBM 650	IBM Corporation	£ 70,000+
IBM 701	IBM Corporation	
IBM 702	IBM Corporation	
IBM 704	IBM Corporation	£ 600,000
IBM 705 I	IBM Corporation	£ 533,400
IBM 705 II	IBM Corporation	£ 533,500
IBM 705 III	IBM Corporation	£ 540,000
IBM 709	IBM Corporation	£ 866,700
IBM 832	IBM Corporation	
IBM 1401	IBM Corporation	£ 120,000
IBM 1410	IBM Corporation	£ 150,000+
IBM 1440	IBM Corporation	£ 30,000
IBM 1620	IBM Corporation	£ 35,000
IBM 1620 Mark II	IBM Corporation	
IBM 1710	IBM Corporation	
IBM 7010	IBM Corporation	
IBM 7030	IBM Corporation	£1,500,000
IBM 7034	IBM Corporation	
IBM 7040	IBM Corporation	£ 305,500
IBM 7044	IBM Corporation	£ 350,000
IBM 7070	IBM Corporation	£ 450,000
IBM 7072	IBM Corporation	£ 400,000
IBM 7074	IBM Corporation	£ 450,000
IBM 7080	IBM Corporation	£ 840,000+
IBM 7090	IBM Corporation	£1 million
IBM 7094	IBM Corporation	£1 million
IBM 7701	IBM Corporation	
IBM 7750	IBM Corporation	
IBM 7950	IBM Corporation	
IBM 8000	IBM Corporation	
ICT 1200	International Computers & Tabulators Limited	£ 25,000
ICT 1201	International Computers & Tabulators Limited	£ 33,000
ICT 1202	International Computers & Tabulators Limited	£ 45,000
ICT 1300	International Computers & Tabulators Limited	£ 45,000
ICT 1301	International Computers & Tabulators Limited	£ 65,000+
ICT 1400	International Computers & Tabulators Limited	
ICT 1500	International Computers & Tabulators Limited	£ 75,000
ITT 7300 ADX	International Telegraph	£ 266,700
KA 21	Standard Elektrik Lorenz	£ 40,000
KDF 6	English Electric Co.	£ 60,000
KDF 9	English Electric Co.	£ 120,000
KDN 2	English Electric Co.	£ 20,000
KDP 10	English Electric Co.	£ 400,000
Kiev	Academy of Science of the Ukrainian Soviet Republic	
KL 901	Societe Nouvelle d'Electronique	
Leo I	Leo Computers Ltd.	£ 95,000
Leo II	Leo Computers Ltd.	£ 95,000
Leo III	Leo Computers Ltd.	£ 200,000
L-3060	General Precision, Librascope Div.	
Librascope ATC	General Precision, Librascope Div.	
Librascope 500	General Precision, Librascope Div.	
M1, 2, 3	Made in Russia	
M20	Made in Russia	
Madam Mk I	Manchester University	£ 40,000
Madam Mk II	Ferranti Ltd.	£ 45,000
Maddam	Burroughs Corp.	
Magloc 1	Sperry Gyroscope Co. Ltd.	
Maniac II		
Mercury	Ferranti Ltd.	£ 120,000
Merlin	Brookhaven Inc.	
MESM	Ukrainian Academy of Sciences	
Metrovick 950	Metropolitan-Vickers Electrical Co. Ltd.	£ 20,000
Micropac	R.C.A.	
Miniac	Marchant Calculations Inc.	£ 28,400
Minsk 1 & 2	Made in Russia	
Mobidic	Sylvania Electric Systems	
Monrobot Mk VI	Monroe Calculating Machine Co.	£ 33,700
Monrobot IX	Monroe Calculating Machine Co.	£ 5,000
Monrobot X	Monroe Calculating Machine Co.	£ 14,200
Monrobot XI	Monroe Calculating Machine Co.	£ 12,500
Monrobot MU	Monroe Calculating Machine Co.	£ 250,000
Narec	U.S. Naval Research Laboratories	
National 102	National Cash Register Co.	
National 107	National Cash Register Co.	
NCR 303	National Cash Register Co.	£ 50,000
NCR 304	National Cash Register Co.	£ 285,000
NCR 310	National Cash Register Co.	£ 32,000
NCR 315	National Cash Register Co.	£ 120,000
NCR 390	National Cash Register Co.	£ 25,000
National Elliott 405	Elliott Bros. Ltd.	£ 120,000
National Elliott 405M	Elliott Bros. Ltd.	£ 130,000
OKI	OKI Electric Industry Co. Ltd.	
Omega 203	Olympia Werke AG	£ 44,500
Oracle	Oak Ridge and Argonne	
Ordvac	University of Illinois	
Orion	Ferranti Ltd.	£ 300,000
Packard Bell 250	Packard Bell Electronics	£ 20,000
Packard Bell 350	Packard Bell Electronics	£ 40,000
Packard Bell 440	Packard Bell Electronics	£ 30,000
Panellit ISI 609	Panellit Ltd.	
PDP 1	Digital Equipment Corp.	£ 60,000
PDP 3	Digital Equipment Corp.	£ 58,700
PDP 4	Digital Equipment Corp.	£ 17,400
Pegasus 1	Ferranti Ltd.	£ 50,000
Pegasus 2	Ferranti Ltd.	£ 120,000
Perm	Technische Hochschule	
Perseus	Ferranti Ltd.	£ 150,000
Philco 1000	Philco Corp.	£ 83,000
Philco 2000/210	Philco Corp.	£ 533,000
Philco 2000/211	Philco Corp.	£ 666,000
Philco 2000/212	Philco Corp.	£ 840,000
Philco 2400/410	Philco Corp.	£ 120,000
PICO	Honeywell	
Pluto	Ferranti	
Prodac 510	Westinghouse & Univac Division of Sperry Rand	
Prodac 580	Westinghouse & Univac Division of Sperry Rand	

Name	Manufacturer	Price
Poseidon	Ferranti	
Rand Johniac	Remington Rand Univac	
Raycom	Datamatic Corporation	£ 85,000
RCA Bismac	Radio Corporation of America	£ 500,000
RCA 301	Radio Corporation of America	£ 100,000
RCA 501	Radio Corporation of America	£ 300,000
RCA 601	Radio Corporation of America	£ 650,000
RCA 604	Radio Corporation of America	
Readix	Idaho-Maryland	
Recomp II	North American Aviation Inc.	£ 31,700
Recomp II	North American Aviation Inc.	£ 21,700
Royal Precision LGP 30	Royal McBee Corp.	£ 18,000
Royal Precision RPC 4000	Royal McBee Corp.	£ 29,200
Royal Precision RPC 9000	Royal McBee Corp.	£ 40,000
SA 100	Wayne Kerr Corp.	£ 1,750
SDS 910	Scientific Data Systems	£ 14,000
SDS 920	Scientific Data Systems	£ 30,000
SDS 930	Scientific Data Systems	
SEA 3900 (CAB 3900)	Societe pour l'Exploitation des Procedes SEA	£ 105,000+
Setun	Made in Russia	
Siemens 2002	Siemens & Halske	£ 100,000
Sirius	Ferranti Ltd.	£ 17,000
Stantec Zebra	Standard Telephones & Cables	£ 28,000
Strela	Made in Russia	
Storekeeper	Electronic Machine Control Ltd.	£ 4,750
Sylvania 9400	Sylvania Electronic Products Inc.	£ 900,000
TAC (Marconi)	Marconi Ltd.	£ 10,000
Teleregister Telefile		
Tosbac 3100	Tokyo Shibaura Electric Co.	
Tosbac 4200	Tokyo Shibaura Electric Co.	
TR 4	Telefunken GmbH	£ 300,000
TR 5	Telefunken GmbH	
Trice	Packard Bell Electronics	£ 250,000
TRW 33	Thompson Ramo Wooldridge Inc.	
TRW 130	See AN/UYK 1	
TRW 230	Thompson Ramo Wooldridge Inc.	
TRW 300	Thompson Ramo Wooldridge Inc.	£ 50,000
TRW 330	Thompson Ramo Wooldridge Inc.	
TRW 400	Thompson Ramo Wooldridge Inc.	£ 666,000
TRW 530	Thompson Ramo Wooldridge Inc.	
Univac File Computer 0 & 1	Remington Rand Univac	£ 100,000
Univac 1	Remington Rand Univac	£ 426,700
Univac II	Remington Rand Univac	£ 500,000
Univac III	Remington Rand Univac	£ 333,000
Univac 120	Remington Rand Univac	£ 32,700
Univac 422	Remington Rand Univac	£ 16,500
Univac 490	Remington Rand Univac	£ 350,000+
Univac 1101-1105	Remington Rand Univac	£ 500,000
Univac 1107	Remington Rand Univac	£ 833,400
Univac 1218	Remington Rand Univac	
Univac Larc	Remington Rand Univac	£1,800,000
Univac Larc II	Remington Rand Univac	£2 million
USS 80/90	Remington Rand Univac	£ 66,700
USS 11	Remington Rand Univac	£ 130,000+
USSC-STEP	Remington Rand Univac	£ 113,400
Ural 1	Made in Russia	
Ural 2	Made in Russia	
Ural 4	Made in Russia	
Verdan	North American Aviation Inc.	
Wegematic 1000	Aktiebolaget Addo	
X 1	N.V. Electrologica	£ 110,000
ZRA 1	Zeisswerke GmbH	£ 40,000
Zuse 11	Zuse KG	£ 10,000
Zuse 22	Zuse KG	
Zuse 23	Zuse KG	£ 33,000
Zuse 31	Zuse KG	£ 40,000
Zuse 64	Zuse KG	£ 20,000
Zuse 80	Zuse KG	£ 21,000

APPENDIX III—LIST OF AND SHORT NOTES ON CALCULATORS USED IN EUROPE IN 1963 (SPECIAL ONE-OFF MACHINES, OF WHICH THERE WERE MANY, ARE NOT INCLUDED)

Name	Manufacturer	Price
Bull Gamma 3	De La Rue Bull Machines Ltd.	£ 10,000
Bull Gamma C33	De La Rue Bull Machines Ltd.	£ 3,800
Bull Gamma G172	De La Rue Bull Machines Ltd.	£ 7,000
Bull Gamma G300	De La Rue Bull Machines Ltd.	£ 11,500
Bull Gamma G322	De La Rue Bull Machines Ltd.	£ 7,500
Deciplex	Southern Instruments Ltd.	
Deciplex K1011	Southern Instruments Ltd.	
Deciplex K1012-A	Southern Instruments Ltd.	
Deciplex K1013	Southern Instruments Ltd.	
IBM 602	IBM United Kingdom Ltd.	£ 5,000+
IBM 602A	IBM United Kingdom Ltd.	£ 5,000+
IBM 604	IBM United Kingdom Ltd.	£ 7,500+
IBM 609	IBM United Kingdom Ltd.	£ 14,000+
IBM 626	IBM United Kingdom Ltd.	£ 6,000
IBM 628	IBM United Kingdom Ltd.	£ 17,500+
IBM 632	IBM United Kingdom Ltd.	£ 2,500
IBM 644	IBM United Kingdom Ltd.	
IBM 3000	IBM United Kingdom Ltd.	
ICT 542	International Computers & Tabulators Ltd.	£ 8,100
ICT 544—EMP	International Computers & Tabulators Ltd.	
ICT 547—EMP	International Computers & Tabulators Ltd.	
ICT 548—EMP	International Computers & Tabulators Ltd.	
ICT 549—EMP	International Computers & Tabulators Ltd.	
ICT 550	International Computers & Tabulators Ltd.	£ 13,200
ICT 550/2	International Computers & Tabulators Ltd.	£ 13,200
ICT 555	International Computers & Tabulators Ltd.	£ 25,000
ICT 556—PCC	International Computers & Tabulators Ltd.	£ 20,000
ICT 557—PCC	International Computers & Tabulators Ltd.	
ICT 558	International Computers & Tabulators Ltd.	£ 15,100+
Univac 1004	Remington Rand Limited	£ 20,750+

COMPUTERS AND PEOPLE

Societal Concerns
The Computer Profession
Issues in Computing
Applications Serving People

The U.K. privacy white paper 1975

by A. S. DOUGLAS
The London School of Economics and Political Science
London, England

ABSTRACT

It is now some years since the formation of the Younger Committee, set up to study the question of Privacy and its protection as it related to the private sector. The Committee published its report in 1972. Shortly after the Younger Committee was set up, an interdepartmental working party was set up within central Government to study the problem as it related to the public sector. This working party's report has now been issued along with a Government White Paper, which makes specific proposals for action in the computer field. In the meantime other countries, including the U.S., have taken significant steps in the direction of defining Privacy and introducing data protection procedures.

INTRODUCTION

The White Paper has been promised for some eighteen months, but there have been successive delays and it has long been an open secret that disagreements over its content have existed in official circles.[1-2] The final version[3] issued owes a great deal to Alex Lyon, the Minister of State at the Home Office, who sat on the Younger Committee and whose minority report is incorporated in the report of that Committee's findings. The White Paper was drafted by a lawyer, Mr. Paul Sieghart, who spent considerable time in wide-ranging consultation beforehand under the aegis of the Home Office and no little time in subsequent amendment to take account of comments and objections!

In discussing the various reports and the White Paper it is essential to distinguish clearly between Privacy, Confidentiality and Security. These three concepts are often confused. However, there are important differences involved, not only in the topics covered, but also in the attitudes to them which can and should be adopted by the professional community, and in the acceptability of those attitudes to the general public. For a brief but clear statement of what is involved see "The Application of Computer Technology for Development."[4] The question is discussed at more length in the Younger Report and elsewhere.[5]

WHAT IS MEANT BY 'PRIVACY'

It is sufficient here to point out that Privacy concerns the rights of an individual, personal or corporate, to enjoy protection from unwarranted and unauthorised intrusion into his affairs. It is not related in any specific way to computers.

It is a social phenomenon and depends critically on public opinion. In a democracy it is clearly a political matter, to be dealt with by our elected representatives —under the (hopefully!) watchful eye of the electorate. It defines *what* is to be protected, not how this is to be done.

Every professional is a citizen also. In that capacity he has a right to speak on privacy, but only on the same basis as every other citizen—his professional status confers no special virtue on any pronouncements he may make. However, in regard to the legal definition of what is to be protected, or to the methods of protection of data regarded as private, professionals in whose area the matter lie may speak as experts and carry an appropriate degree of authority. This authority carries with it a responsibility to explain the nature of any problems involved—in language laymen can understand—and to warn of the probable results of abuse, misuse or omission of safeguards. It is in this spirit, I shall approach the rest of this paper.

CONFIDENTIALITY

Confidential material is data entrusted to an individual, corporation or agency, whether local, national or international, the possession of which imposes a duty on that agency to protect it from disclosure except as authorised by the protocols under which it may be collected. It may include, but is not limited to data about individuals. The protocols creating confidential status for data may be statutes, contracts or customs recognized by the Courts or by common consent. The protocols sometimes conflict with other protocols, when the individuals involved may well be placed in an invidious position. A journalist asked to reveal his sources when these are connected with criminals, for example, may feel that he must preserve the anonymity of the sources partly to preserve them from retribution and partly to ensure that he is able to use similar sources at a later date—yet his action could be deemed to conflict with the criminal law and place him in jeopardy as an accessory to a crime. In some cases the conflicts have been resolved by the Courts in favour of preserving confidentiality except in specified circumstances—e.g., the doctor-patient and solicitor-client relationships, where the preservation of confidentiality is generally acknowledged to be in the public interest unless criminal proceedings are involved.

The protection of confidential material is, of course, closely related to the preservation of privacy in respect of data about an individual. The security considerations are similar. But in the case of privacy the rules under which disclosure may be made are not necessarily well defined, if they are defined at all. Since there is no right to privacy defined in English law, rules of disclosure must currently rest solely on any protocols which may exist or be created by, e.g., contract. It is interesting to note that this is not so under the Code Napoléon, on which French law is based, in which the point is covered to some extent by the 'droit de la personalité'. (For a comparative study of the position under different laws, see Reference 6.) A political move is now afoot in the U.K. to cover this 'open' position by Statute. It will be interesting to study the White Paper which will in due course, no doubt, emerge.

It is fair to say that the White Paper on Privacy discussed here does not directly define Privacy or Confidentiality but primarily addresses itself to the question of Security. However, there is one important point on which it is clear. This relates to the question of disclosure of the existence of data banks containing records concerning individuals. The White Paper proposes that disclosures should here be the rule (para 34 & 35) and that exemptions on grounds of national security would need a certificate from the Home Secretary (para 39).

SECURITY—SOME GENERAL REMARKS

There are two points to bear clearly in mind when examining any security system. The first is that there is no such thing as a perfectly secure system. In general, one gets the degree of protection one is prepared to pay for—and perfect protection has infinite cost. It is generally cheaper to design features into the system from the start than to add them later. The aim must be to ensure that penetration of the system costs at least as much as the possession of the data is worth to a penetrator, and to design the system so that penetration, if it occurs, is detected.

The second point is that the weakest link in any security system is the people involved in it. Almost all breaches of security involve the co-operation of someone inside the system. Thus the selection and control of personnel is a key issue in providing protection of data. Moreover, only those 'inside' a system are normally in a position to detect certain types of abuse, before these effect the public. Thus their cooperation is essential.

These points are implicitly recognised in the White Paper, and specific mention is made of the role that the British Computer Society could play in relation to any legislation which may follow from its provisions (para 43). But the mechanism involved has been left open for the present. We shall discuss this further below.

The essential elements of a security system must involve legislation, detection of breaches, and enforcement. The White Paper, whilst proposing legislation, is not definitive on the method to be used to detect breaches or by whom the law is to be enforced. However, one important aspect on which it is clear is that the mechanism of enforcement should apply equally to the public and private sector. The delay in prohibiting the Report of the Interdepartmental Working Party which completed its work in 1972, indicates that this view may not have been wholly shared by the official side. To understand this, it is perhaps necessary to review existing practices.

THE OFFICIAL SECRETS ACT, 1911

In the public sector, protection of data has been practised for many years, of course. Indeed, specific duties of protection are laid down in various statutes to preserve the interests of the individual. Cross-correlation of data relating to an individual has been controlled by a set of internal rules drawn up within the Civil Service. It is a fairly sophisticated code and takes account of the need to obtain valid statistics, whilst at the same time preserving the anonymity of the individuals whose records are used in those statistics. It is recognised that any such procedure can lead to a risk of disclosures about individuals, but there is insistence on this risk being minimised by the techniques employed.* It is fair to say that a specific case of unauthorised disclosure has yet to be proved, although it is not difficult to 'manufacture' situations in which it could take place. To this extent existing protection is to be commended as being effective and sensible.

Enforcement of the rules is by administrative fiat. The unauthorised disclosure of data by an individual Civil Servant not covered by any other protocol is covered by the Official Secrets Act,[7] under which every Government Servant who handles any confidential or private data operates. Signature of a declaration under the Act is not essential to its effect being binding on

* For some valuable comments on this aspect of matters see 'Report of a Working Party on Survey Research and Privacy' 1973, issued by Social & Community Planning Research, 16 Duncan Terrace, London N1 8BZ.

persons involved in handling such data, but helps to draw attention to its provisions. The Act is comprehensive and was drawn up with matters of national security in mind. There are thus "catch-all" clauses which can be extended to cover matters which might be considered to be secret only in a national emergency—and then only by the stretch of a very fertile imagination. It is, for example, possible to bring under the Act the important topic of whether and how often the grass is cut at a designated Government installation. Whilst one can undoubtedly construct an ingenious scenario in which such a matter is of crucial national importance, one can reasonably feel that, in normal circumstances, it is unlikely that this is of great concern to anyone—friend or foe—except perhaps an aspiring gardening contractor!

Use of the Act to support a case against reporters and others concerned with the Nigerian civil war led to some public concern a few years ago. There have been rumbles about its effect ever since, and a Departmental Committee set up in 1972 under the Chairmanship of Lord Franks recommended the repeal of Section 2 and its replacement by an Official Information Act.[8] But any alteration is naturally met by an equal concern about the effects which could flow from its repeal and a general uncertainty as to what amendments are really needed.

There are, no doubt, those who deem the Act necessary in some form, as I do myself. Many of them, like me, feel that it is a somewhat blunt instrument as it now stands. It is administratively convenient, because it confers wide powers to impede the flow of information both inside and outside Government on senior civil servants when they deem it necessary. Whilst I have no reason to doubt the sincerity of their intentions in the public weal, nor can I point to specific instances of abuse related to computer matters, the system could all too readily be abused, and there is at least circumstantial evidence that it is, from time to time, used more to cover up incompetence than to protect what must be protected for the good of all. To this extent the present system could usefully be amended.

The White Paper, if implemented, must necessarily impinge on the working of the Act in so far as it relates to computer-supported data bases concerning records of individuals. Since non-disclosure on grounds of national security of the existence and nature of such a data base would require the certificate of the Home Secretary, if the provisions of the White Paper become law the provisions of the Act could then only be used if that exemption were to be granted. This implies a review procedure which is not necessary at the moment. Furthermore, the mechanisms for enforcement considered by the White Paper both imply that, if complaints are made concerning the effects of the existence or use of such a data base to the enforcing authority, that authority will need to know of the existence of such an exemption order, and could ask for it to be reviewed by the Home Secretary if it felt that exemption was being abused.

While there are those who would welcome the abolishment of the Official Secrets Act or, at least, its substantial modification, it is encumbent upon them, in my view, to state how those of its provisions deemed essential to national security are then to be dealt with. For the most part, this has not been satisfactorily tackled by would-be amenders. Total abolitionists would not, I think, obtain a Parliamentary majority, and would undoubtedly be opposed by a majority of officials.

A partial answer might well evolve from the legislation proposed by the White Paper. Clearly the review mechanism introduced as described above would tend to reduce the unselective use of blanket security provisions, and a sensitive use of his powers by the Home Secretary could lead to greater openness overall, by insisting on a careful definition of what features of a system actually attract exemption, initially in relation to computer-based systems, but, in due course, by extensions to manual systems. It is to be hoped that this evolution will indeed take place.

ENFORCEMENT—THE PROPOSED DATA PROTECTION AUTHORITY

The White Paper describes two possible forms of enforcement agency, which it tends to contrast. The one with a tighter form of control would involve a process of licensing. The White Paper does not set out to determine whether this relates solely to the licensing of the organisation holding the data, the collection agency, the people involved, or all three. Nor is any specific mechanism proposed for ensuring the existence of a 'responsible person' in relation to the data, although this point is discussed at some length in the Younger Report. Any or all of these forms of control are possible under the suggested provisions, however.

The "alternative" seen by the White Paper envisages an agency which only reacts in response to a complaint of an abuse of the citizen's rights by virtue of the existence or operation of a data base. The proposed Data Protection Authority in this case would act only to investigate such complaints and take action to remedy them where they proved to be justified.

An interim Data Protection Committee is being set up to make recommendations on the form the Data Protection Authority should eventually take. The terms of reference of this Committee are not spelled out in the White Paper directly, but, by implication, include advice on the legislation to be enacted. They will almost certainly be required to define, at least provisionally, some of the individual's rights to privacy as well as spelling out the powers and duties of the Data Protection Authority. It is, perhaps, significant that Sir Kenneth Younger has been appointed the Chairman of this Committee. In the light of the work done by his earlier Committee, we may expect that the definition of privacy will follow the lines of its Report, modified by any impact made by inclusion of public sector considerations.

COMMENTS ON ENFORCEMENT

The problem of enforcement is perhaps best seen in relation to a real-life situation, albeit a hypothetical one. U.K. firms are currently subject to a Pay Code, forbidding the paying of increases in wages in excess of a fixed figure each twelve months. It is true that this does not exactly have the full force of criminal law, but a breach is certainly punishable, if detected. In all firms which use their computer for payroll one or more programmers are authorised to see the figures paid out, although these are regarded as confidential. The programmers concerned are usually asked to sign papers drawing their attention to the confidential nature of their work and under law can be dismissed by their employer for a breach of such confidentiality.

Let us suppose that one such programmer notices a breach of the Pay Code in the course of his duties, what should he do? He can, of course, report it to his superior. However, more likely than not he will be told to forget it and reminded of his obligations relating to confidentiality. He can resign, but with present employment prospects, he might find that, at the least, inconvenient. A threat to resign in order to reinforce his report to his senior, is clearly a dangerous course, bordering on blackmail at worst, and almost certain to rebound on him in due course. He can consult his professional Society—assuming he belongs to one. The Society can operate at a higher level than he is able to do, and stands some chance of getting the situation put right, but there is no way to avoid the finger pointing at a small number of people as containing the one responsible for the 'leak'. Naturally straight dismissal would be out of the question as the adverse publicity the Society would ensure would not be welcome. But it is difficult to see how some retribution could be avoided if the management were so minded. The least that would be likely would be a transfer to other work.

Clearly the dice are loaded against disclosure of the breach coming from anyone, however professional and public spirited, inside the system, unless they are on the point of resigning anyway. Moreover any tangible proof of their contention which could be produced by them would constitute theft of company property—all the more culpable if they took it with them on leaving.

The only possible counter to this seems to be to strengthen the 'outside' pressure on the individual professional to act according to a code of ethics and good practice and to protect him, if he does so, from reprisals by his employer. An enforcement authority would need to be equipped with wide licensing powers to do this, applicable equally to the conduct of operations and the personnel involved. It is also clearly practically desirable to have a named person to accept responsibility if things go wrong, who is empowered under the licensing procedure to enforce proper control even though this may not be in the direct interests of his employer. The parallel drawn in the Younger Report is to a mine safety officer, who is employed by the mine but has statutory powers and duties in respect of safety. The parallel is not exact, but illuminating.

It is, I think, clear that a 'complaints board', however strongly armed with investigative powers and sanctions on future action, would be unlikely to achieve a significant change in employee attitudes. But this is not to say that some complaints procedure is not desirable for a licensing authority. I do not myself see the White Paper proposals as strictly alternative. To an extent they are complementary. It is arguable, obviously, that it is possible to have a complaints handling procedure without licensing. However I do not think that, in practice, a licensing procedure will work smoothly without a complaints mechanism.

The British Computer Society, in addressing the Home Secretary on the subject, has come out firmly in support of a licensing authority (see Appendix). One form of such a system is already in operation in Sweden, under their Data Act.[9]

COSTS

The White Paper refers specifically to costs, and it is clear that the Data Protection Committee, in making its recommendations, will need to justify them on a cost/benefit basis. The Swedish licensing system does not involve a large central staff—25 at present, I believe, and direct costs are covered by charging a licensing fee. Some of the factors involved in implementing recent legislation in the U.S. are set out in a recent Rand Corporation report.[10] A similar study will be needed to assess the costs, both direct and indirect, incurred in implementing the alternatives proposed in the White Paper. Whilst, as in the Swedish system, central costs can be kept down and funded from license income, if this route is chosen, there are bound to be additional costs at each installation. This raises, in particular, the question of consultation of records by those about whom they are kept. Although the right to inspection is advocated in principle in the White Paper it is clear that the cost of this will need to be reviewed carefully and some arrangements set up that will enable the individual to be assured of the correctness of data held about him without either he himself, the data bank operator or the public purse being put to unreasonable expense. It is here that proper advance planning of the data base system for filing and retrieval in the light of security and accuracy requirements will have its most important role to play. A well planned system which satisfies a reasonable licensing requirement may well be little more expensive to construct and operate than one which does not. The main problem of cost is likely to arise in respect of existing systems, which cannot readily be replanned to serve the new requirement. Clearly, temporary measures may be necessary to deal with this problem. More study is certainly needed before costs can be determined.

Clearly the costs of a complaints system must also be carefully considered. At first sight it would seem that the burden will have to fall on the taxpayer, since charging the complainants would rapidly defeat the intentions of the system by discouraging complaint and a system of fines on firms found at fault used to fund the system might lead to over-vigorous pursuit of complaints if the 'quota' of fines had not been received.

In fact the method of financing has a deeper significance than a narrow question of cost/benefit of operations; it goes to the root of the independence of the Authority itself. Any arrangement depending for its finance on taxpayer's money can be influenced by the funding agency—no doubt in this case the Home Office. This is plainly undesirable if the Authority is to act even-handedly in both public and private sectors. It would, of course, be possible to insulate the members from influence by giving them the Status of Her Majesty's Judges who can only be dismissed by a vote in both Houses of Parliament—but this is unlikely to be considered appropriate at this stage. An institution dependent on license fees which it can levy as of right to recover its costs seems the more independent of the alternatives available. Financial independence is specifically commended in the White Paper.

PERSONNEL SELECTION AND LICENSING

Another important question, should the Authority become an established part of our national life, will undoubtedly be who appoints the members of the Authority. There is growing concern being evinced in the Press and Parliament about the nature and extent of Government patronage. This probably seems somewhat surprising to those who live in a country where rather more Government posts are filled by political patronage than in the U.K.—and perhaps we are rather too touchy on the subject. The concern is, I think, less over the influence exercised by Ministers themselves than over that being increasingly exercised by the Civil Service. To pretend that this does not happen in all bureaucratic systems would be hypocritical, but to condone an unnecessarily arbitrary and subjective practice would be equally so. The process of advertisement and competitive interview used in the Universities and in recruitment for the Civil Service itself still seems to be the most objective procedure available, but this is not used in practice for appointments to such bodies as the Data Protection Committee or the proposed Authority. I certainly hope the Authority will not emerge as a self-perpetuating body setting up its own system of succession. Obviously careful thought needs to go into this question.

So far as the licensing of personnel operating data banks is concerned, I have explained above why I believe it is important to 'control' the people involved as well as the organisation keeping the data bank. Here "control" can be assisted by the existence of a national professional body or bodies whose standards include a code of ethics and a code of good practice attuned to the needs of the situation. Whether or not the societies concerned maintain rigid restrictions on entry to qualified persons is perhaps less important than their members should accept the need for a degree of responsible discipline in their work. The Swedish Computer Society for example whilst subscribing to the latter concept has yet to embrace the former and may well never do so. But I have no doubt that its existence is of value in operating their Data Act.

The B.C.S. would not, I believe, wish to seek a monopoly under any licensing system, but would certainly seek to ensure that its membership at an appropriate level would carry with it automatically a license to practice at a licensed installation, and this would undoubtedly simplify and strengthen the implementation of a licensing scheme.

CONCLUSIONS

The White Paper has gone a considerable way in the direction pointed by the Younger Report, and, in particular, to satisfying the various concerns expressed by the British Computer Society in its submission to the Younger Committee, as developed subsequently by the Privacy and Public Welfare Committee of the Technical Board. As I have indicated, there are a number of matters still to be defined and worked on before an Act is finally introduced—indeed, more than I have had time to develop here. The various Committees of the B.C.S. within the Technical Board are already at work on these with a view to making a submission to the Data Protection Committee, with whose work they hope to be even more closely associated than they were with that of the Younger Committee. I am proud to be associated with this effort, which, I feel is helping to maintain in the U.K. the high standards of liberty and open Government for which we are renowned. I am also very grateful to the organisers of this Conference for giving me the opportunity to present this report to those who share our concern that computer systems should be used to the advantage and not to the detriment of individual liberty.

REFERENCES

1. Report of the Committee on Privacy, 1972, H.M.S.O. Cmnd 5012.
2. Computers: Safeguards for Privacy, 1975, H.M.S.O. Cmnd 6354.
3. Computers and Privacy, 1975, H.M.S.O. Cmnd 6353 (the White Paper, i.e., a statement of Government Policy).
4. *The Application of Computer Technology for Development,*

Second report of the Secretary-General, 1973 U.N. Sales No. E.73 II.A.12 para. 50, p. 73. See also First report of the Secretary-General 1971 U.N. Sales No. E.71.II.A.1 paras. 214-7, pp. 73-4.

5. Gotlieb, C. C. and A. Borodin, *Social Issues in Computing*, Academic Press 1973, particularly Chapter 5.
6. *International Social Science Journal*, XXIV, 3, 1972, pp. 413-602, published by Unesco. See also Ref. 2, paras 69-85. For the position in France see 'Report of the Commission on Liberties and the Computer to the Lord Chancellor, 1975, Documentation Française.
7. Official Secrets Act 1911, available from H.M.S.O.
8. Report of a Departmental Committee on S.2 of the Official Secrets Act 1911, 1973, H.M.S.O. Cmnd 5104.
9. Swedish Data Act 1973; for commentary in English on its provisions see *Current Sweden*, Swedish Institute No. 4, July 1973.
10. Turn, Rein, *Cost Implication of Privacy Protection in Databank Systems*, 1975, Rand Corporation, p. 5321. See also EDP Analyser for November/December 1975, particularly December, p. 11, and 'The Cost of Privacy,' Goldstein, R. C., 1975, published by Honeywell Information Systems Inc.

APPENDIX—BCS POLICY ON COMPUTERS AND PRIVACY

Preamble

The BCS has 22,000 members from all elements of the computing community, by whom it is guided and influenced. As such it is aware of the existing and potential impact of computing technology on the personal and economic lives of us all.

The BCS recognises that computing techniques have much to offer in improving the general quality of life of both the individual and the community. Computing technology is an asset of such power for assisting us to attain real wealth that it becomes essential for it to be used effectively.

Any source of power may be used or misused—the issue of privacy and computers is concerned with ensuring that control may be exercised in such a way as to avoid and prevent the misuse of personal data and in so doing also to encourage the beneficial use of such data.

Computing within our community has attained a stage at which legislative control is required as the White Paper of December 1975 acknowledges.

The policy

1. The policy of the BCS is to encourage the effective use of computing for the benefit of individuals and society and we believe that all regulatory measures to be introduced should reflect this positive attitude.
2. There is a need for legislation to control the use of personal data held on a computer.
3. The regulatory measures should apply to both the public and private sectors.
4. Regulation should be carried out by licensing.
5. Regulation should be associated with appropriate enforcement procedures.
6. The Regulatory measures should include a complaints and appeals mechanism available to organisations and the general public.
7. Regulation should involve the approval of the original purpose and any subsequent revisions thereto for which personal data is collected and processed.
8. The Licensing Authority should have the duty to disclose the existence and purpose of collections of personal data unless it is satisfied that substantial reasons exist to the contrary.
9. Regulatory measures should include the definition of the responsibilities and duties of individuals engaged in a licensed function affecting personal data, together with the authorisation of those individuals.

The Society believes it to be desirable that the Regulatory Authority should be wholly independent and preferably self-financed. However, although data processing costs are currently falling the expenses of the regulatory measures to be embodied should be arranged so as not to inhibit the wider exploitation of computers.

Human and organizational implications of computer privacy

by JERRY M. ROSENBERG*
Polytechnic Institute of New York
Brooklyn, New York

ABSTRACT

The computer was created to serve man. With increasing reason serious questions are being raised pertaining to the potential misuse of the computer. Specifically, the issue of privacy has come to the public's attention as cherished liberties and psychological needs are gradually being eroded as the technology of the computer expands.

Recent discoveries have shown how men desiring power can utilize data for their own benefit, leaving the person intruded upon with little recourse, or for that matter, any knowledge of the invasion into his private world.

This paper identifies the many implications of loss of privacy on man's ego, dignity and general ability to cope with these forces. Failure to possess privacy can lead to withdrawal, loss of productivity, lack of confidence, physiological deterioration and other conditions of mental and physical decline.

A significant portion of this paper details the protections that can be incorporated, primarily technical, to preserve the balance between the need for society to know and the need of the individual to remain independent. The basic conditions for man's well-being are being tested along with the foundations of our governmental traditions.

At present, computers have an almost limitless capability to store, intermingle and, at the push of a button, retrieve information on persons, organizations and a variety of their activities, all without the knowledge of those involved. Even now, stacks of punched cards and tapes store statistics about us that we may not know exist. We might never escape in time or distance the bureaucratic machinery keeping tabs on us.

With present technical capability, it is possible to develop a composite picture of an individual that can be stored in a single information warehouse. Each year we offer information about ourselves which becomes part of the record. It is often scattered across the continent and is usually inaccessible except after considerable effort. It begins with our birth certificate and is followed by a series of medical notations. Early in life we are documented as an added income tax deduction by our parents. Then there is information on what high school, public or private, and what college, public or private, we attended. At school, records are made of our abilities, grades, tests of intelligence and attendance. For some, there will be car registration and driver's license, draft status, military service or Peace Corps. Then job history is recorded —working papers, Social Security number, a first job, our performance with each employer, recommendations, and references—all this makes an interesting dossier. Then perhaps, a marriage license, a home mortgage, and when children come, the cycle begins anew. Should we divorce, the court records will be added. These would increase should we be arrested, convicted or serve time in prison. And of course, when we die, a last footnote is made.

In our daily activities we leave behind a trail of records: the credit card carbon for a luncheon meeting, the receipt from the hotel where we spent last night, our airline ticket, the check we cashed in a city bank, and the bill for the toys we charged for our children.

There are also government dossiers including tax returns over a number of years, responses to census questionnaires, Social Security records, passport files, and perhaps, our fingerprints and military intelligence reports. If we have worked for a defense contractor or for the federal government, there are lengthy files on us that may note our associations and affiliations.

Information is power. These records may at various times be of considerable interest to people inside and outside a specific government agency. Years after our birth, for example, an interested party may be happy to pay for information from our birth certificate which is officially confidential. And in a number of cities there are entrepreneurs who obtain and sell this in-

* Dr. Jerry M. Rosenberg is the author of "The Death of Privacy —Do Government and Industrial Computers Threaten Our Personal Freedom?" (Random House, 1969). *This paper is based on testimony before the Senate Judiciary Committee on Constitutional Rights.*

formation as well as hospital records, police records, immigration records, and so on.

Confronted with the erosion of his privacy, the individual American has until now had the consolation that all these files have been widely dispersed and often difficult to put together. It has been a time-consuming, expensive proposition to compile a sizable file on any individual. Giant computers with their capacity for instant recall of a great variety of available information are changing all this.

The evolution of computerized complexes without effective public participation and protest can have a serious impact on our democratic process. Under our present system, individuals are expected to make fundamental choices where the future welfare is at stake, as would be the case in an election. By alienating the people from the decision-making process, control of the computer technology is left in the exclusive hands of those in possession of organizational power.

The public itself should question the drift of these technologies. We should want to make certain that human dignity, psychological well-being and civil liberties remain intact. We should demand to know the precise nature of the information that will be stored and who will have access to it. The public has the right to know who will have the power to control the computers and most importantly, how confidentiality and individual privacy can and will be protected.

Liberty is never gained once and for all. It is forever in conflict with civilization—a conflict which has no clear-cut solution but which reappears in cycles, usually in different forms. Each succeeding generation must win it anew. Each must defend it against ensuing dangers. This is necessary because we are constantly changing our life environment; society may be altered so frequently that safeguards that in the past adequately protected our liberties become obsolete.

Science and technology are of immense benefit to society. These advances are so important to us that we would not want under most circumstances to impede their movement in advancing our knowledge of the world. But they may also expose us to potential danger—to a pollution that could curtail our anonymity, solitude and privacy. Unless certain practices in the technological exploitation of scientific knowledge are restrained, they will cost us more than we should be willing to sacrifice.

And we must constantly evaluate these technologies which are tools developed to increase man's power to understand his world. The mere fact that an innovation presents itself does not mean that we should surrender years of experience and values to its authority. Yet it is difficult to bring social pressure to bear against the control of potentially dangerous technologies. One reason is that those who have the use of the technology are influential enough to prevent societal, or for that matter, legal restraints.

That privacy will forever remain because it is implied in the Constitution and Bill of Rights is not credible to the new adult population. With growing hostility toward the dominating technology and the establishment, a segment of this group fear that the documentation of their so-called acts of rebellion will only show that the freedoms once assumed have been surrendered. Should our older citizens in power fail to come to grips with the issue of preservation of privacy, it can be expected that the last struggle will be made by those who question how their present behavior, if documented, could be used against them at some future time.

The computer cannot be blamed for the loss of privacy. It is but an instrument created by man. Computers and other advanced machine systems are not permitted to be in error, but man is not a machine and does not have to be as efficient as the tools he has created to serve him. If man loses his right to be wrong, will he react by withdrawing from society? Will his curiosity to experiment with life falter? If this happens, man truly becomes nothing more than a machine.

Of course, not all computerized systems contain potentially damaging information. Some operations merely act as accounting systems and high-speed calculators, while others at more sophisticated levels are depositories for internal decision-making; some store research information from diverse sources; and some are documentors for the purpose of assimilation and distribution of pertinent data to a large community.

Not all computerized complexes contain the "sensitive" or potentially "threatening" information that might be found in a computerized system designed to collect personal data. But the possibility of incorporating such information does exist. Even the rather elementary, antiquated computer has the potential for being an information storage center. It doesn't matter whether it is formally called a "bank" or a "single unit processor"—any capacity to collect, store and retrieve data instantaneously upon request may, if misused, infringe on personal privacy.

There is little doubt that as computerized communications systems spread throughout the nation and world, surveillance by data processing is bound to increase. If the trend continues, it will soon be possible to have all personal information about an individual gathered on a continuous basis and held indefinitely until requested. The snowballing effect is quite pronounced here. When the decision is made to purchase a computer, more data are gathered about the employees, customers or taxpayers who are of interest to an organization. Although this may provide for better services, improved decision-making and policy-programming, it also provides personal information about individuals never known before the advent of computers.

ALONG THE ROAD TO PSYCHOLOGICAL SUBMISSION

Today man lives in an atmosphere dominated by the machine. He brushes his teeth with an electric toothbrush; prepares his meals with mechanical toasters, ovens and broilers; works in an atmosphere of motors, switches, fans, typewriters; goes to and from home by car, bus and train; reduces the chores of home life with sewing machines, washing machines and drying machines. In the past only the craftsman used the tool. Today all of us take machinery for granted. As long as machines served us and did not threaten our rights as persons, we welcomed technology.

The charm of the horse-drawn buggy yields to the modern automobile; the candlestick maker is not needed in this day of electric power; the complexities of the abacus are incorporated into the computer's memory unit. Often we are glad to say goodbye to what we leave behind because many innovations free man from monotony, physical effort and waste of energy.

Computers are part of this advance, aiding us in ways that are valuable for our everyday living and essential for progress on all levels. Much of what has been achieved in medical research and outer-space exploration would have been impossible without the wide range of sophisticated computers.

Unfortunately, sacrifices frequently accompany these changes. With all the splendid wonders of the computer we find ourselves asking: has man become submissive to the computers of today? Can each individual profess to be more human in his actions than the complex system he has developed to assist in daily endeavors? Will there be a growing tendency to create a world where we treat each other as machines? Are we building more barriers which prevent the individual from having the opportunity to evolve his own unique potential—to be self realized?

Man submits more and more as his ability to make choices about and control his future is gradually taken away from him. He is willing to have the machine make numerous decisions for him about his future; he is willing to permit the machine to build towers of brick and metal, hoping that it will not fail him when he has to live or work in them; he is willing to have the machine process his life's facts, hoping that it will be accurate and objective.

It seems that we are not aware of what is happening to us—that we are losing a little each day to the machines. We are usually too busy to think about matters which seem on the surface not to be so "important" as whether our cars are safe, or the price of bacon or the way taxes are skyrocketing.

What is most disturbing to the American population is the undemocratic process which starts at birth to make people believe that they are unable to say "no" to divulging personal information, thus perpetuating a collection of data that will follow them for the remainder of their lives—"frozen in time and the computer."

People want to determine for themselves in every particular situation of life just how much of their complex beliefs, attitudes and actions they choose to disclose. To the American, this data is more than just statistics. It is the data of judgment, a possible last judgment that can affect their schooling, employment possibilities, promotion, or role in the community. The citizens of this country want to have the right to a personal diary that is away and free from the organization's outstretched hands. They plead the case that if all their actions were documented, including their mistakes, it would be difficult to close a page of one's life and start anew. It would be a tyranny over mind and destiny.

To maintain their dignity and fill their need for psychic distance, people construct mental walls around themselves. To be a total psychic being, with stability and confidence, forces people to reject being intruded upon without permission. Psychologically, privacy demands a delineation of the self, the acceptance that each of us is unique and separate from all others. It recognizes an empathy toward the finer qualities within man. It demands the perpetuation of a private psychic domain, displaying a defensive shield against psychological penetration, unless authorized.

There is a growing antagonism against people desiring power, who will through mental coercion try to intrude upon our concealed thoughts. Unfortunately, we have learned that the man who wishes to gain control will employ various techniques to influence and force individuals and groups into submission.

People have a right to remain unique and different. But there are many, and indeed the number is growing, who intentionally or by title of their office, are against the solitary man. They may envy his uniqueness. They want to keep a close watch on his behavior so as to anticipate future moves, often defended in the name of science or national priority. They too often regard his privacy as a denial of their own mechanized psychology which has a stereotyped and oversimplified answer for everything.

COMPUTER COMMUNICATION AND PRIVACY PROTECTION

A major problem in protecting our privacy is that too often we believe in the principle that the ends justify the means. When we consider that the goal is the greater good of our people, we cannot understand why a specific intrusion should be prohibited. The result: gradual erosion of the value we place on individual privacy. Sometimes we are confused and become easily convinced that a particular device that may lead to personal intrusion is warranted on other grounds, such as purposes of security. This is an inadequate argument I believe.

As computer networks spread throughout the country and world, science and privacy must be able to thrive together. We will be collecting thousands of facts about everyone, depositing these details into the unforgetting computers of the future. To date there are no adequate legal protections to safeguard the individual against computer leakage. Furthermore, laws alone will not offer satisfactory protection in the face of widespread use of these systems. Although laws can impose penalties for violation and can set the limits of proper safeguards, legislative actions have not always been effective in the control of surveillance activities like wiretapping and eavesdropping.

There is reason to hesitate before passing new legislation that might in fact backfire. Laws that give special agencies or departments the responsibility of investigating those who break the law would be introducing yet other bodies that decide who can know what, thus putting a new decision-power in the hands of a few.

We have to make sure that information given to a specific organization will not be shared in such a way that the person's identity will be discovered. It is necessary to specify those who may use certain technological devices. Neither the principal of a school nor a personnel director should be allowed to enter at will the dossier on a potential or present student or employee. The question of duration of surveillance is most important. In addition, we need to determine what kinds of electronic devices are appropriate and permissible.

We must define the penalties that would be imposed on those who disclose information improperly or without authorization, and we must regulate the use of information for purposes other than those for which it was originally obtained.

We must also bear in mind that we are dealing with a super-technology that will become increasingly complex and difficult to evaluate. It is safe to assume that probably the only persons who will understand the complexities and operations of these systems will be the computer designers and systems engineers who are directly responsible for the evolution of the industry.

Safeguards can be inserted into a system in use, but it would be more efficient and less costly to build them in at the time the computer is designed. The burden of a great deal of the responsibility must lie with the computer manufacturers. If they want to avoid external regulations, they will have to start thinking about how to design systems with built-in safeguards.

To date, the best attempt to identify the relationships between computer surveillance and invasion of privacy has been outlined by Petersen and Turn of the Rand Corporation. They visualize two types of disclosures of information—accidental disclosures resulting from failure of the computer, and deliberate disclosures from infiltration of the system. They suggest countermeasures to prevent surveillance of data within a computerized system.

Unfortunately, essential safeguards are not as easily attained as is suggested by some of these outspoken specialists. It is one thing to design countermeasures as they apply to the "general" concept of computer leakage; it is quite another matter to build in protections for a specific computerized system.

For example, few can find fault with Petersen and Turn's countermeasures but they are merely a theoretical framework for the complex changes that are needed. These countermeasures offer little assistance to those attempting to design a surveillance-proof computerized system in the medical field, in an educational community, for a corporation or for a government repository. Examples of a specific computer utilization within a defined framework are necessary. The rules that apply for one computer installation might be inadequate for another or might fail to respond to the more crucial or pressing needs.

There are certain general rules of conduct pertaining to *all* computerized data centers that should be followed in order to increase confidentiality and reduce information leakage:

1. Let people know what their records contain, how they are used and protected, and who has access to them.
2. Employ a verification process to insure accuracy of data; in addition, permit the individual to review the data for accuracy, completeness, current application, and freedom from bias.
3. Categorize all stored information as intimate, private and therefore non-circulating (such as physical, psychiatric and credit information); pertinent, but confidential and having limited distribution; or public, and therefore, freely distributed.
4. Regard personal data as personal property, requiring permission for its use, and punishment for its improper use.
5. Appointing an ombudsman agency—or a committee that represents all levels of the organization—to take major responsibility for hearing and responding to complaints, and to determine appropriate measures to minimize leakage.
6. Record each request for access that is made, along with the authorization.
7. Make security checks on computer personnel.
8. Assess, from time to time, people's attitudes toward and anxieties about the issue of invasion of privacy. Such studies could be useful in determining what form of records would be most acceptable.
9. Periodically review and update the adequacy of the physical safeguards. Employ capable outside consultants to attest to the safety of the

systems used, and to assist in the development of appropriate technical devices (such as scrambled data and code names), and

10. Allow psychological seclusion and withdrawal from accountability to remain as a permanent stronghold of our value system. The individual must freely choose whether or not he wishes to become submissive to the power of the computer.

A creative response by the computer industry to its technology will probably serve, and satisfy, the public better than rewriting our laws. In fact, one can doubt that legal measures—although necessary—will be as effective as technological adjustments in the protection of the public's privacy.

What is needed before the establishment of large computerized centers is a rigorous research effort to answer the following unresolved questions:

1. What are the purposes of a computerized central facility? What kinds of information are strictly relevant to these purposes?
2. How much information about an individual is required to guarantee that such services are useful to the person, community and nation? How accurate, objective and challengeable is the information?
3. What are the procedures for the sharing of the system?
4. How will individuals be protected from the creation and distribution of derogatory data caused by clerical mistakes or computer malfunction?
5. Will procedures be developed to permit individuals to see their files?
6. Will the cost of such a facility be justified in terms of future savings?
7. Will there be adequate safeguards to prevent penetration from the outside?
8. In whose backyard should computerized centers be physically established?
9. Will a computerized center officially created as a statistical system eventually become a storehouse of personal information? and
10. Does the concept of computerized communication centers suggest a changing value system and further intervention in the lives of Americans?

The burden of proof of the security of the data facility should lie primarily with those who propose it. They must demonstrate that they can create a virtually impenetrable and incorruptible system and justify its greater economy and expanding service.

The dialogue has just begun. The right to preserve privacy is a right worth fighting for. Events of the past several years clearly point to the need for a redesigning of our communications protections. Computerized systems offer great potential for increased efficiency; yet they also present the gravest threat of invasion of our innermost thoughts and actions. Transactions of our personal movement will glut the records and offer a very up-to-date picture of how we conduct ourselves in private. Some see this trend as leading to an Orwellian nightmare with Big Brother watching over us and reporting to the central record-control authorities any behavior adjudged out-of-line with stated policy.

We are slowly drifting into a world of nakedness. Each year an increasing number of technological devices invade the world that once we considered private and personal. In spite of this, we are still confident that our lives, activities, ideas, thoughts, and sensations are shared with no one unless we so choose. Will this confidence be perpetuated? Echoes of Watergate, CIA spying, domestic surveillance, wiretapping, etc. may well have shattered any future acceptance by the public. An uphill effort is required.

The snowballing effect of computers is very real indeed. The more you know, the more you want to know and the better your methods will become to get and integrate this information. In the end, will there be any place to hide?

Computers may continue to prove themselves the worthy servant of man. But the servant must yield to his master, and the necessary thought must be given to developing essential safeguards. The computer manufacturers have thus far shirked their responsibility, but they cannot long remain bystanders if they wish to continue to make their own decisions. Both the manufacturers and then the consumer must seek ways to control the all-documenting, all-remembering computer systems and demonstrate that machine technology need not necessarily bear the stamp of increased surveillance.

The ultimate submission must be of the machine to man. If we fail to act immediately to preserve our claims to anonymity, psychological independence and seclusion we may develop a permanent fear—a fear to enjoy the fuller opportunities of life. We will hesitate before experimenting with the challenges of the world. We could become carbon copies of one another—conforming, dull and psychologically equivalent to the computer—heartless and non-emotional.

A control systems model of privacy

by JOHN SALASIN
The MITRE Corporation
McLean, Virginia

ABSTRACT

Concerns about individual privacy, specifically in relation to automated data systems containing personal information, are considered in terms of a feedback control system model. This model provides a framework which appears to relate many separate concerns which individuals or groups have expressed about privacy. It also provides a framework which may be suitable for analyzing legislation or regulations promulgated for the protection of privacy. The model may assist in defining needed research on the need for privacy, or on the impact of inadequate protection of privacy.

The model posits that data systems pose various "threats" to privacy depending on the extent and manner in which the existence of such information systems hinders the ability of individuals or groups to provide feedback to systems which affect them.

INTRODUCTION

A recent workshop addressing "The Privacy Mandate"[1] reached consensus that "our current lack of understanding about specific needs for, and feasibility of, implementing comprehensive privacy laws indicates a need for continuing research in many areas." One of the areas of research suggested was the "privacy needs and desires of individuals as affected by situations, culture, and economic level." Comments from the floor, following presentation of workshop findings, included the statement that:

> *"Proposed legislation seek to deliver and protect a 'right of privacy.' Nowhere is such a right identified or defined. We recommend that present efforts be redirected, at least in part, towards the development of a clear and complete identification of the nature of privacy as a concept, perhaps several concepts, depending on varying situations."*

This paper presents a model which may be useful in developing an "identification of the nature of privacy," at least as the concept of privacy applies to the use of information systems containing personal data. The concept of privacy employed in developing the model is broader than that of information security. It is, rather, based on Westin's view that privacy is the claim of individuals, groups, or institutions to determine for themselves when, how and to what extent information about them is communicated to others.[2]

The model incorporates concerns which individuals have expressed about computerized data systems containing personal information and regulations which have been promulgated to address these concerns in the framework of a feedback control system.

The model attempts to separate concerns about the privacy impact of computerized information systems from inherent characteristics of such systems. These inherent characteristics include, for example, potential reductions in the cost of processing large amounts of data, the potential for updating and checking the accuracy of files more frequently (or for increasing errors due to improper coding and transcription), and the use of more "objective" (e.g., coded) information. These "characteristics" do not, in themselves, necessarily raise concerns about privacy. The model is based on the premise that such concerns can be explained (and studied) by viewing automated personal data systems as control systems, rather than through examination of such inherent system characteristics.

The material presented is based largely on testimony presented before the Secretary's Advisory Committee on Automated Personal Data Systems and on literature cited and recommendations made in that Committee's report.[3]

A CONTROL SYSTEMS DESCRIPTION OF INFORMATION SYSTEMS

Control systems can be roughly categorized as being of two types. The simplest type of control system employs "external" control, or control unresponsive to the output of the system being controlled. Many control systems which utilize personal data have traditionally been of this type.

The credit reporting industry, for example, controls the issuance of consumer credit by providing data to indicate an individual's eligibility to receive credit.

Until passage of the Fair Credit Reporting Act, however, individuals affected by the system were often not aware of its precise nature. They could object to the fact that credit had been denied, but had little opportunity to provide feedback to the control portion of the system (i.e., the particular credit bureau providing information). The difficulties experienced by individuals who have attempted to correct erroneous bills from gasoline companies or other credit-granting organizations indicates that someone affected by the system may not be able to provide input to the control system which determines, in this case, his ability to purchase goods using the credit instrument.

A second type of control system employs "feedback control," or a control which is determined partially by the output of the system being controlled. An automated oil refinery serves as an example of this type of control mechanism. External controls, such as market demands, availability of crude petroleum products, etc., are used to determine an "optimum mix" of output products. An attempt to operate the system considering only these external factors could, however, lead to a major explosion at the oil refinery, since the output products of the refinery must also be related to the current state of the chemical processes involved. While running the refinery without external controls would be uneconomical, running without feedback control based on the temperature and pressure of ongoing reactions would be catastrophic.

Feedback in the control of social systems can be taken to represent the ability of individuals or groups affected by a system to respond in a manner which might modify the behavior of the system. In the case of an information system containing personal data, this response is not a primary input to the system itself but, more accurately, a feedback signal to the control portion of the system. Any loss of feedback is, in this control system approach, equivalent to eliminating the potential for input from the persons affected.

While most physical systems allow a relatively clear distinction to be made between input and feedback signals, such differentiation may become difficult in social systems since both input and feedback information is often provided by the same person or group. In some state welfare systems, for example, eligibility is determined primarily from a declaration provided by the applicant. Such information could appropriately be considered as input. Feedback might take the form of an individual's objection to a decision reached in his case, or in a larger sense, societal reactions to the management of the welfare system. Feedback is thus characterized as representing input generated in response to prior system output. We emphasize that elimination of this feedback is not an inevitable consequence of automation. It can be avoided if care is taken in the design and operation of systems.

There are two types of concern about automated data systems. First is concern by an individual who is directly affected by the system (e.g., a welfare recipient), and therefore is highly concerned about his inability to provide feedback. Second is the concern of individuals who, though not directly affected by the system, are concerned about the institution's effectiveness in serving the interests of the public.

POTENTIAL HARMFUL EFFECTS OF AUTOMATED PERSONAL DATA SYSTEMS

An individual's feeling that he is unable to provide feedback to a system affecting him may reflect seemingly rigid and impersonal treatment of him by organizations maintaining data systems. This feeling can be heightened by the very nature of computerization. Computers are programmed by supplying specific rules for operations to be carried out on the data being processed. While any well-designed system includes provisions for processing both the normally expected data and all foreseeable exceptions, the data used for input must conform to pre-established rules. It is, for example, often expected that numerical values will be within a given range or that an individual's name will consist of alphabetic characters only. While such expectations are often reasonable, they do impose artificial limitations. The existence of rigid procedures may make it necessary for individuals to interact with the system according to *its* rules. This is, in itself, a block to providing feedback to the system.

An individual's feeling of inability to provide feedback may also be created by a system's reliance on records, as opposed to personal contacts, in making decisions about individuals and groups. Such reliance on records may be inappropriate if the record used to make a decision does not contain enough information or have enough reliability to be the basis for a "just" decision. Relying on records for decision-making will, because of economic necessity, place a much greater emphasis on stored, and hence possibly old, information. It is evident that a greater reliance on records is aided by the fact that the computer has made the records more readily accessible, often cheaper to obtain, and therefore more convenient to use than information collected and verified by personal contact. The economics of computerization makes it possible to rely on computers for low-level decision making (e.g., initial screening for job candidates) when dealing with large populations of individuals on whom one has readily accessible information.

As a side effect of this reliance on records, individuals are concerned that unequal treatment may accrue to various segments of the population because of the extent to which their lives are documented by automated records. The comprehensive record-keeping practices of such institutions as the military, psychiatric hospitals, and correctional institutions combine with the fact that poor, black, Spanish-speaking, and unskilled individuals are more likely to be drafted, have a criminal record, have a mental illness or deficiency treated in a state institution, be on welfare, and enroll in a government retraining or special education program. Thus these individuals will be trailed by a far greater volume of possibly derogatory computerized records than most middle or upper class Americans.

There is a diminished opportunity for individuals to "make a new start," free of the records of their past activities. The greater accessibility of more data, much of it obsolete (due to an increased storage, linkage, and retrieval capability without sufficient provision for expunging old records) leads, as was previously noted, to a greater institutional reliance on records. If there are no legal constraints on informal exchanges of data and on the amount of time that data can be maintained in a file, the danger is increased that institutions will use old and possibly inappropriate records in reaching decisions.

The two effects mentioned (i.e., rigid treatment of individuals and reliance on records) might cause other effects which contribute to the concern of individuals over their inability to provide feedback to systems. The rigidity of a system may lead to distorted interpretation of data by the compression, standardization, and oversimplification of data elements. This fitting of people to codes rather than codes to people leads to misinterpretation of the data. Thus, while a police blotter might show that an individual was arrested "for refusing to supply positive identification on request," the National Criminal Information Center (NCIC) may be informed, because of the required standard code, that the individual was charged with "resisting arrest." Poorly designed systems might be characterized by the statement that: "It can't be done if there isn't a code for it." People appear less likely to have feedback worries if they feel able to present "their side."

Individuals are also highly concerned that organizations will make decisions about them without their knowledge. People fear unwarranted, peremptory, or discriminatory institutional responses to non-conforming behavior which is detected covertly using computer analysis. They are concerned that institutions will use personal data contained in data banks to detect and respond to behavior considered (by the institution) to be undesirable. The ability of the computer to rapidly retrieve and analyze large masses of data facilitates "browsing" through records in an attempt to determine relationships among the data. For example, computers might be used to identify a small percentage of families in a given city that account for substantial portions of the serious crime and/or welfare caseload so that discriminatory action might be taken against these families. The significant element to this concern is that there is no opportunity for the individual to provide feedback to affect these actions.

As more social functions become controlled by data systems, it becomes harder for an individual to register objections about the fact of inclusion in a specific system. To participate in society we are required, depending on our age, class, education, etc., to be included in files of students, drivers, members of a particular profession, patients at a hospital, or other files pertinent to our activities. A student in a migrant farm worker's family appears to have little choice regarding whether or not his records are maintained in the "Uniform Migrant Student Transfer System." Even if he, his parents, or his teachers were able to provide feedback regarding the accuracy or completeness of information stored, could feedback be provided which registered his objection to being labeled a "Migrant Student"? While the computer may not have caused society's desire for information, it has made it more feasible to use this information in the control of social processes. The resulting pervasive nature of such computerized information systems, combined with the rigid classification procedures often applied, tends to limit the manner in which individuals can provide feedback to control systems employing the technology.

A feeling of individual powerlessness is, to some extent, contributory to the second type of concern about the use of automated personal data systems; that the use of such systems by institutions may weaken the institution's effectiveness in serving the interests of the public. If individuals do not feel they can have an impact on organizations making decisions about them, they may not believe the organization is fulfilling its social mission. This feeling may be evidenced in increasing distrust, fear, or frustration of individuals and groups in regard to institutions; these feelings may be promoting a notable reserve or hostility in dealing with record-keeping operations. If the process of computerization in a large organization is not visible to the individuals affected; the individuals may feel, correctly, that they are denied opportunities to provide feedback to the organization.

There is also growing skepticism about the effectiveness of the institutions in meeting their own organizational goals. The reduction of feedback can cause such skepticism by reducing the ability of individuals and groups to contest the acts of institutions, which, because of their control of large data bases, may claim to have "the true facts," "more persuasive evidence," or "more comprehensive view" of an issue. Distortions or misinterpretations of data which reflect assumptions underlying the design of the data system might result in policy ill-suited to the goals of the organization.

The tendency of institutions to "blame it on the computer," combined with the specialization inherent in computerized operations, may result in a diffusion of institutional and individual accountability for decisions and their effects. It becomes difficult for either an organization or an individual affected by the organization to assign responsibility for decisions arising from erroneous data or programs. For example, an error could occur because the input data were not correct, because input data represented a special case which the computer program was not designed to handle, or because the output data were interpreted erroneously.

Related concerns arise over the unintentional erosion of the autonomy and authority of institutions. As economies of scale cause institutions to pool data bases and computer hardware, it is difficult to determine which agency has the responsibility for maintaining the data or the right to access certain portions of it. Such an erosion of autonomy seems particularly significant in systems which combine police and court records or in data bases shared by Federal and State governments, since combination of these systems may result in a blurring of the separation of powers. It is currently common practice to allow private insurance companies access to drivers' license records maintained by State governments; such practices might weaken the distinction between the public and private sectors.

All of these effects create a concern that social institutions employing automated personal data systems are out of human control. The individual feels that he is unable to provide feedback relating to either the goals or method of operation of an institution.

LONG TERM SOCIAL EFFECTS

Feedback is generally designed into physical systems to provide stability in the face of varying inputs. The higher the "gain," or amplification factor, of a system is, the more necessary that feedback be employed to prevent small disturbances in the input from causing major perturbations in the output. Automated personal data systems, having the ability to process more records more rapidly than manual methods, might be thought of as processing a higher gain (i.e., variations in the system itself, or systematic changes in input data, may have a greater effect on a larger number of people).

In addition to the immediate effects of the misuse of automated personal data systems, any widespread feeling of individual powerlessness in affecting institutional behavior may result in major social changes. Such changes might include: increased social isolation accompanied by hostility between those responsible for social institutions and those served by them; increasing skepticism about the effectiveness of institutions, leading to a state of anarchy as the institutions cease being able to function; or the establishment of a new "ethic" which, contrary to western (American) tradition, negates the concepts of "free will" or "self-determination." While it seems improper for us to make value judgments respecting the nature of society in future generations, we have an obligation to prevent such changes from occurring without review. Individuals should be allowed the opportunity to provide feedback to the process of social change. If particular elements of automated personal data systems might induce such changes, people should be able to provide feedback which could modify those elements.

FEEDBACK CONSIDERATIONS IN THE PROTECTION OF PRIVACY

Just as the concerns of individuals about privacy can be expressed in terms of their ability (or inability) to provide feedback to systems, so regulations promulgated to protect privacy can be analyzed in terms of the ways in which the regulations foster or inhibit feedback from individuals or groups.

Information systems might be considered in two categories for determining types of feedback appropriate to a system. The first category includes those systems used for entitlement or classification of individuals, thus directly affecting the file subject. Examples might be credit files, school records, medical files, or police records. The second category, "statistical systems," includes systems which may not affect the individual file subject directly but are used by organizations for planning and evaluation. The decennial census of individuals in the United States is an example of such a system, since these data are used extensively for policy decision making.

Statistical systems may or may not contain sufficient information to allow data to be linked to specific individuals. Those which do not contain personal identifying information (e.g., name, address, social security number, etc.), cannot easily be used to directly affect individual file subjects. If such identification is maintained however, information in the system can be used in a manner which directly affects the individual. If a given system is used both for making decisions about individuals and for statistical purposes, several forms of feedback may be required.

Many of the factors which influence the design of feedback control to physical systems may also be applicable to ensuring feedback to information systems. It is, for example, extremely important that the output effects being used to determine the feedback control are, in fact, the "right" outputs. If the system being controlled is not understood completely, the relevant outputs may not be discernible. A simple example might be that of a husband and wife sharing a double bed with an electric blanket that has separate controls for each side. In this hypothetical situation, assume that the controls have been switched inadvertently, so that the control on one side of the bed controls the temperature on the opposite side. Considerable difficulty arises if the wife attempts to turn up the control on her side of the bed; the husband begins to get too warm and turns down his control. The wife, realizing she is not getting warmer, turns the control up further; how long the procedure continues depends on the tolerances of the individuals involved.

The problem in this situation is that inappropriate output is used to generate a feedback control signal.

Each person is providing feedback based on the temperature on his or her side of the bed, rather than on the temperature of the opposite side, which is actually what each controls. The appropriate output for each to react to would be the spouse's comment that one was either too hot or too cold. Adjusting the control based on this verbal input rather than on the temperature of one's own side of the bed would better insure that an appropriate control was applied. In a complex social system it becomes even more difficult to be sure that all relevant output effects are included in an appropriate feedback loop.

We are concerned not only with what output effects are measured, but also how they are measured. If effects are measured periodically, rather than continuously, we run the danger of being unable to tell what the true output is. Shannon's sampling theorem states that if a varying signal, or wave, is measured by sampling at time intervals, the sampling rate must be at least twice as great as the highest frequency of variation in the original signal if we wish to reproduce the signal. If we were to assume that bicycle theft varies cyclically, with a periodicity of one year and a maximum occurring in July, any conclusions which were reached about bicycle theft in this country might be biased if measurements were taken only in the month of July. An erroneous picture of system output can make any feedback employed ineffective.

The output effects measured and the method by which they are measured contribute to the effect which feedback has on the system. Control science differentiates between "positive" and "negative" feedback. While negative feedback will tend to dampen oscillations in the system, and thus promote system stability, positive feedback may accentuate such oscillations and lead to system instability.

The delay inherent in a feedback loop can have a major impact on the effect of feedback. A feedback signal which normally acts in a negative manner and, therefore, improves system stability may, if an appropriate delay is chosen, create system instability. While it becomes difficult to make firm analogies between a physical system and a far more complex and variable social system, we might consider court processes to represent a potential delay in social systems. If each feedback action initiated by an individual or group experienced a two-year delay while awaiting judicial action, the delay might be capable of causing system instability.

FEEDBACK ANALYSIS OF PRIVACY REGULATIONS

Recommendations made by the DHEW Secretary's Advisory Committee on Automated Personal Data Systems,[3] many of which were incorporated in the Privacy Act of 1974, can serve as an example of how regulations to protect privacy can be analyzed in terms of a feedback control system model. Thus, for example, the Committee recommended that individual data subjects be provided the following rights:

(1) To be informed of his (or her) right or privilege to refuse to provide data about himself and of the consequences of such refusal. Such notification should allow an individual to make an informed decision about whether or not he wished to be included in a given system. This most fundamental type of feedback, allowing an individual to decide about his own inclusion in a system, is essential to enabling feedback.

(2) To be informed (upon request) that he is the subject of a record contained in a system and to obtain the data contained in the record. Individuals must be aware of both their inclusion in a system and of the information stored about them if feedback is to be possible. While informing (automatically) an individual that he is a data subject without his requesting this information would go further towards facilitating individual feedback, the Committee found that such a requirement might be "needlessly burdensome to some organizations."

(3) To have assurance that data about him are not used for purposes other than the stated purposes of the system without his informed consent and to inform him (upon request) about the uses being made of the data. An individual should be allowed to provide feedback regarding the use of information about him. Individuals should be allowed to provide feedback relating to dissemination (and additional uses) of information about themselves; both because the dissemination itself may affect them and to alert them that another organization may be making decisions about them based on the data.

(4) To contest the accuracy, completeness, or pertinence of a record or data, and to alter the record or data as necessary to assure its accuracy, completeness, or pertinence (i.e., mechanisms will be established to allow feedback).

These requirements are designed to allow individuals to provide feedback regarding whether or not they wish to be included in a system; the accuracy, completeness, or relevance of data contained about the individual in the system; and potential uses of the record both inside and outside of the system.

For those systems defined as "statistical," whether or not personal identifying information is included, the ease with which an individual can assure himself of the accuracy, completeness, or propriety of a record about him may be less significant. Random data errors in individual records might not affect the statistical uses to which the data are put as long as the error rate was reasonable. The feedback capabilities provided for such systems must allow interested individuals or groups to assess validity of the decisions made on the basis of the records. The following might be required of such statistical systems:

(1) as in the case of systems directly affecting the individual, the individual should be informed of his right or privilege to refuse to provide data about himself;

(2) the individual should be informed that the information collected from or about him is to be maintained in a data base used for statistical analysis. The individual should be informed of the policies governing institutional responses to subpoenas or other requests for information contained in the data base;

(3) provision should be made to allow review by any interested party of all techniques used in arriving at a decision based upon analysis of data contained in the data system. Such review might include evaluation of the sample selection, data collection instruments or procedures, information storage policies, analysis techniques, and any other elements of the data analysis process;

(4) data used by the organization in making the analysis should be made available, in machine-readable form, to any individual or organization desiring to undertake independent analysis of the data. Individual personal identifiers should be removed from the data before such transfer occurs and, if due to sample size, sampling procedures, or other characteristics of the data base, a likelihood exists that individuals might be identified even in the absence of personal identifying information, data should be aggregated to a sufficient degree to make such identification impossible;

(5) the organization maintaining the data should be responsible for preventing transfer to, access to, or use of any data in personally identifiable form by any person or organization, including staff of the organization itself. Data collected for statistical purposes should not be used in a manner which could affect the individual directly.

The management requirements which might be placed on organizations maintaining data systems containing personal information are similar regardless of the type of system. They are designed to ensure that the organization provide for appropriate feedback and that, if information is collected for statistical purposes, that the information is not used in a way which would directly affect the individual file subjects.

Any organization maintaining an automated personal data system might be required to fulfill the following general management requirements:

(1) designate one person as responsible for the operation of the system and for assuring that appropriate feedback mechanisms are provided;

(2) take affirmative action to assure that each of its employees having responsibility for the design, development, operation, or maintenance of the system, or for the use of data contained in the system, is informed of all requirements relating to the operation of the data base and of all of the rules and procedures of the organization which are used to ensure adherence to the requirements;

(3) provide for sanctions to be applied to any employee whose willful or negligent conduct causes or significantly contributes to the denial of feedback to an individual or group;

(4) provide for sanctions to be applied to any employee who initiates any disciplinary or other punitive action against any individual who brings to the attention of appropriate authorities, the press, or any member of the public, evidence that opportunities for feedback have been denied to an individual or group.

The above examples of recommendations for the protection of privacy can all be viewed as guaranteeing that individuals will be able to provide feedback to systems which make decisions based on files containing personal data. Other potential recommendations, regarding the design of systems and the requirement that the public be notified of the system's existence, can also be viewed as protecting the right of individuals to provide various types of feedback to systems which employ personal files as part of their control element.

POTENTIAL MODEL UTILITY

The feedback control system model of privacy, as outlined in this paper, provides a framework which may help coalesce many of the separate concerns which individuals or groups have expressed about privacy. Similarly, the framework appears to provide a unified framework for analyzing the effectiveness of privacy legislation and regulation. The model can also provide guidance to those responsible for developing and operating information systems—does a system provide for adequate feedback from those affected by the system?

The model may provide a basis for conducting research on the need for privacy, or on the impact of inadequate protection of privacy. Individuals might be surveyed, or requested to participate in hypothetical scenarios, to determine, for example:

- the types of feedback which individuals feel are required for different types of systems (thereby providing a categorization of systems in terms of feedback required).
- the relative value of different forms of feedback

(e.g., feedback regarding accuracy vs. feedback regarding dissemination).

- potential "tradeoffs" which might be made between different forms of feedback provisions and other costs (or benefits) to data subjects.

Finally, a more fully developed model of privacy as a feedback function should provide guidance for the ongoing monitoring of privacy protection mechanisms.

Any assessment of the important system output effects or of the appropriate feedback mechanisms might prove, over time, to be somewhat in error. It is often difficult, even in purely physical systems, to determine exactly what output measures are appropriate to use for feedback control. The fact that output measures and feedback mechanisms chosen might be partially incorrect, or that the appropriateness of feedback forms could change with time, does not obviate the necessity of providing such mechanisms. The pervasive nature of regulations for the protection of privacy, combined with the fact that the public should be encouraged to use whatever feedback mechanisms are to be provided, makes it advisable that the types of feedback provided, and the implementation of regulations which ensure the feedback, should be continuously reviewed.

Ongoing review of feedback procedures might include monitoring organizations' compliance with regulations designed to facilitate feedback. Public perception of the availability and utility of feedback opportunities should be reviewed.

Regulations established to provide feedback should be evaluated continuously. Further knowledge of the processes involved may indicate the need for changes in procedures, regulations, etc. Social scientists might discover that recommendations have not, in fact, considered the most important output variables with respect to the stability of institutions or society. The mechanisms provided for feedback might act contrary to what is expected. Since most automated systems used today are highly complex and poorly understood, the continuing study of the efficacy of the feedback mechanisms employed is required.

REFERENCES

1. *The Privacy Mandate; Planning for Action, Symposium/ Workshop* National Bureau of Standards and The MITRE Corporation, Washington, D.C., April 2, 3, and 4, 1975.
2. Westin, A. F., *Privacy and Freedom,* Atheneum, New York, 1967.
3. *Records, Computers, and the Rights of Citizens,* Report of the Secretary's Advisory Committee on Automated Personal Data Systems, U.S. Department of Health, Education, and Welfare, July 1973, DHEW Publication No. (OS)73-97.

Computer security—A survey

by PETER S. BROWNE
General Electric Information Services Business Division
Rockville, Maryland

ABSTRACT

With the growing requirements for protection generated by legislation such as the 1974 Privacy Act, the increasing complexity of computer and data communications applications, and increasing awareness regarding computer vulnerabilities, the discipline of computer security is achieving independent recognition. Current data processing literature is a rich source of information. Articles and papers regarding security, design of software protection, operational practices and auditing number in the thousands. Most of them are very narrow in scope or so general that they are of little use.

It is important to the data processing professional to be able to sort out the large body of material in order to gain perspective. This paper attempts that by relying on a carefully selected and fully annotated bibliography of 134 items, many of them of interest to the systems analyst or designer. These papers are referenced in the text, which attempts to carefully distinguish between the technical and operational elements of computer security, while providing an overall perspective.

INTRODUCTION

The computer has unleashed countless opportunities for industrial growth, new applications, labor-saving accomplishments, and improvement of the quality of decisions. Most industrial and governmental organizations could not survive without the processing capability of their computer systems, and it can be shown that society itself is dependent upon the computer.[23] At the same time, computer technology has spawned a whole new field of crime and has generated a series of problems for both designers and users of information systems.[2]

With the growing pervasiveness of computers, their increasing complexity and the development of sophistication regarding computer vulnerabilities, the discipline of computer security is achieving widespread recognition. Many organizations have created the position of DP security specialist or manager[87,92] and college courses in computer security are being taught.[69] There are a number of driving forces behind the interest, some of which are outlined below.

Historical

In the middle 1960's, Congress began discussing the issues of privacy and the computer. A national data bank was proposed. Congressional committees were established, and public testimony published.[132] The general consensus was that that technology had not advanced to the point where privacy could be maintained.

Concern over the inherent lack of controls in computer systems led to much discussion and some activity on the technological front. A landmark meeting of active professionals in computer security in 1972 set the stage for an understanding of the technological issues and led to intensive design efforts to achieve "secure" computer systems.[18]

In the meanwhile, activity on the legislative and social fronts saw a culmination in the Privacy Act of 1974 (Public Law 93-579). This act applied privacy requirements to most computer systems operating within the Federal Government. It also generated a number of papers regarding implementation requirements, [90,105,110] and attempts to determine the true cost of privacy, especially as applied to large, multi-use data banks.[10,57]

The need for computer security is also affected by technological factors. As systems become more complex and sophisticated, so do the problems of data integrity. Resource-sharing systems achieve their greatest advantage when used simultaneously by many customers. This also means simultaneous processing of data with varying needs for confidentiality and pervasive needs for accuracy.[6] The problems of management control also have increased as the flexibility and capability of systems improve.

The scope and complexity of the field becomes apparent when a survey of the literature turns up over a thousand articles dealing with physical security of computer assets, threats to the computer, protection against fraud, embezzlement, and other human failings, the need for insurance, software protection, hardware safeguards, legal considerations, risk assessment, auditing, computer system design and the principles of operating system software security.[1,11,107] A multidisciplinary approach is needed.[95]

Definitions

Computer security is a widely discussed subject, and a generally agreed definition refers to it as protection of data against accidental or intentional disclosure, destruction or modification. Security can be viewed as a problem of "comprehensive control," involving the development of means to insure that privacy decisions are enforced.[37]

Data confidentiality is "the status accorded to data which requires the protection from unauthorized disclosure."[92] It refers to the protection of data from unauthorized disclosure, whether the basis for such protection is agreement, law, policy or prudent judgment.[109]

Privacy is a legal and social concept, having roots in constitutional law and social justice requirements.[66,132] It refers to the right of an individual to control the collection, storing and dissemination of data about himself.[6]

Data integrity is the protection of data against accidental or intentional destruction or modification. It also is the ensuring of accuracy and completeness of data. It involves the need for all components to operate together in a consistent and reliable manner.[133]

It can be seen that the object is data. We have been discussing data security as contrasted with computer security. To include the broader-based definition of the subject, and the need to think of the other assets involved such as computer hardware, facilities and people, the term 'processing integrity' has been coined.[105] It is the property of having adequate processing capability, availability and reliability in order to provide the requisite services of data processing.

PLANNING FOR COMPUTER SECURITY

Threats and vulnerabilities

The result of a security breach is what usually draws attention to a threat, a vulnerability or a particular countermeasure. The short history of computer security is spotted with numerous "horrible examples," fads such as the interest in magnets as a threat, the implementations of security measures that are anything but cost-effective.[80,101,131] A rational approach to the subject implies some sort of quantification of risks, and an analysis of the costs and benefits of countermeasures. Although some articles and papers have called for this approach,[23,30] only recently has there been a serious attempt to model the risk-cost interface.[20,40,79,87]

One of the key steps in devising protection is the classification of various threats. There are two sources of threats, people and natural hazards.[25] It is possible, though not easy, to quantify the threat of fire, earthquake, flood and storm.[79] On the other hand, those events that arise from human acts such as mistakes, disgruntlement, fraud and sabotage are not always possible to quantify, namely because of the complexity of motivations, environmental considerations and the effect of in-place countermeasures imposed.[100] The first step is to organize and classify the threats in a systematic manner.[87] Threats are usually part of the environment. On the other hand, the vulnerabilities of a particular computer system to those threats are dependent on a large number of factors relating to location, people, capabilities of the system, building structure, nature of the processing and operating practices.[79] Most security surveys and evaluations are designed to review these installation dependent vulnerabilities and postulate countermeasures accordingly.[84,102]

Adequate cost-effective protection against data security threats is uncommon. Usually the implementation of computer security is given low priority. It has suffered from inadequate attention and analysis, with too many existing measures lacking flexibility, consistency, completeness and redundancy. These attributes are all necessary in order to achieve protection that works when it is supposed to. One-hundred percent security or reliability is never possible. What is needed is a set of security measures that take into account the failures, errors, omissions and vulnerabilities of any given environment.[23,104]

Risk analysis

Risk analysis is the term applied to the systematic quantification of threats, loss exposures and countermeasure benefits.[20] The ingredients of a risk analysis are the postulation of threats and their probability, the calculation of loss exposures, including degraded productivity, usually on an annualized basis. It is important not to ignore the very low probability, high loss events that occur so infrequently that the annual loss potential appears negligible. A high loss exposure, regardless of the probability, should be evaluated carefully. In any event, the apparent simplicity is misleading. It is not easy to quantify all the potential losses, to postulate all the threats or to estimate their probability. It is also a complex and time-consuming task, which accounts for the relatively few completed risk analyses to date.

OPERATIONAL COMPUTER SECURITY

Computer systems are generally not designed with security as a primary objective.[18] Generally, the large main-frame manufacturers claim that users have been slow to request security. Current research effort by independent sources and manufacturers alike indicate that the next generation of computers will achieve adequate, measurable and certifiable protection in hardware and software.[111]

Much protection for computer systems can be implemented outside of the computer hardware and system software. Managers of computer installations have always been concerned with the problems of system integrity, processing availability and security. For them, physical security, backup and administrative controls are highly relevant.

Physical security

Physical security has been subjected to study and implementation long before the arrival of computers. Implementation of physical access controls to computer facilities represents a generally agreed first step in achieving threat protection. The reason is that many threats, especially of a human nature, can be reduced by limiting access.[28,131] To deal with the threat of fire, utility unreliability and environmental disturbances, numerous control and monitoring systems have been devised. All should be considered in the context of the overall DP security plan, even though responsibility for their implementation may be elsewhere in the organization.

Backup and recovery

Recovery planning to ease the pain if a disaster were to strike is important.[29] The objective is to assess the capability of the organization to respond immediately, and ensure that supplies, data files, programs, documentation and equipment are available off-site. The contingency planning must be of sufficient detail so that in case of disaster, all the elements can be pulled together in order to resume operations in as short a time as possible.[117]

Administrative controls

The administrative burden of proceduralizing and formalizing a security program is generally underestimated. It takes great clerical resources to ensure adequate maintenance of a selective access program, whether it be selective authorization to data files or physical areas. Other administrative aspects include the development and implementation of security policies, guidelines, standards and procedures. Again, these functions may be centralized or decentralized, but stand a greater chance of success if the latter.[39]

Security in recent years has been a major concern of computer operations groups. It is here that the organization can channel resources most effectively to deal with the lack of security in operating systems or in application system design. It is a necessary but not sufficient condition for providing true computer security.[24] One of the best guides for information about secure operating practices is the System Review Manual on Security, published by AFIPS.[102] Other guidance can be found in the more exhaustive of the many checklists and guidebooks on computer security.[44,61,79,84,87,92]

Audit

Audit has been defined as "an independent and objective examination of the information system and its use (including organizational responsibilities) into:

. . . the adequacy of controls, levels of risks, exposures and compliance with standards and procedures

. . . the adequacy and effectiveness of system controls versus dishonesty, inefficiency and security vulnerabilities."[18]

Independent and objective are the key words. Whether or not an auditor's objective is the detection of fraud in computer systems, his role is certainly one of reviewing the adequacy of system security. Many CPA firms have finally recognized their unique role in security assurance.[83,116] Some critics say their attention is still inadequate and not yet relevant.[110] Suffice to say that computer systems need auditing, both internal and external. It is not possible to even consider auditing "around the computer" because of the risks involved. Given the nature of computer related threats and vulnerabilities, the traditional independence and inquisitiveness of the audit profession and the requirement for independent assessment of controls, it is logical that much computer security activity will be a part of the auditor's domain.[26]

TECHNICAL ELEMENTS

Even though the first line of defense is to rely on secure operational practices and physical security, the elements of system design have always intrigued computer security professionals. Obviously things can go wrong with hardware and software. Data integrity, encryption and security surveillance must be considered in any complete computer security program. Understanding of these elements usually requires a person well-versed in systems programming and application system design. That the skills required in this area are completely different (and perhaps incompatible) with the skills required for handling operational security problems has not been well identified in the literature. In addition, no present commercially available operating system is immune from penetration, and so the prevalent attitude is that it is futile to attempt to provide protection against the determined technical penetrator. However, much research and vendor effort is being devoted to the appropriate technical safeguards in operating systems.[111]

Identification

Positive identification of people, devices, programs, systems and processes is clearly a requirement for adequate security. Holding a person accountable for his actions is one of the first principles in good design. This requires certain knowledge that he is who he says he is. There are three approaches to personal identification, (1) identification based on passwords (2) on credit card technology and (3) on personal characteristics of the requestor. Passwords are the most common method, but they suffer from some serious inadequacies.[66] They should be random in nature and of sufficient length to avoid compromise.[6] The use of credit cards, usually with a magnetically encoded stripe, is achieving great popularity, especially in regard to Electronic Funds Transfer Systems. This approach makes sense if the cards are controlled, used in conjunction with a unique personal identifier (PIN number) and if the system is made aware of lost cards so that casual retrieval of a card will not be an open invitation to access. Identification based on personal characteristics, such as voiceprint or fingerprints is still not a commercially popular methodology, but offers future promise.[38] Identification not only relates to personnel access, but also to other system entities. Security objects can be people, terminals groups of people (cliques), programs, terminals, data communications devices or segments of virtual memory.[85] Then one can specify restrictions based on a number of parameters such as the characteristics of the requestor (name, terminal, program, etc.) content of data (all salaries over $30,000), context of data (association of college grades, number of parking tickets and credit rating) or one can use procedures (formularies) based on the nature of the situation.[67]

Authorization

Once a system resource or person is identified, the problem of access of the identified subject becomes an important concern. Authorization refers to the establishment of allowable interactions among system elements.[52,59] The traditional concept of authorization in system design presupposes that any system entity automatically is authorized access to any other system entity unles specifically prohibited. The secure concept of system design takes the opposite view. The concept of "least privilege" holds; namely that any system entity is prohibited from access to another system entity unless specifically authorized. For example, there is no need for a peripheral allocator to be able to control or even have access to user data bases or other elements of the operating system. It should have knowledge of only those resources necessary for allocation of devices to jobs.[18]

The concept of an access matrix espoused by Conway, et al[36] appears to be the easiest way to implement access control, but the implementation is not clean.[59] There are a number of choices that one can make in defining the rules of access. For example, what level or degree of privilege should be permitted? Are we talking about control of access to files, records, elements within records or specific hardware or software elements of the computer system?[52,67,72]

Much of the early work in authorization technology is the result of research activities.[35,42,53] The academic environment has fostered some good studies[55,59] which have led to some actual efforts at implementation. Work at MITRE and the US Air Force on security kernels (provably small security reference monitors)[86] at Stanford Research Institute on proofs of program correctness,[91] at System Development Corporation for the DOD community,[128] at MIT Under Project MAC[112] and at computer system manufacturers,[52,54] has led to actual demonstration of computer and communications systems with security as a prime design requirement. An excellent but dated paper by Saltzer summarizes current (as of early 1975) research and development efforts.[111]

Integrity

Obviously, things can go wrong with hardware and software. Data can be (and frequently is) inconsistent or unreliable. Data integrity interfaces with computer security at almost every point. In fact, many observers see the two concepts as being nearly synonymous.[105] A high integrity operating system can by its nature provide security against unauthorized use of system resources. System integrity is the condition of proper and predictable operation of the total system, including hardware, software and human elements. It includes the physical and operational security mechanisms in place.

Part of the integrity solution lies in providing an operating system that does not treat every operation as "benevolent," but in fact assumes that users are going to attempt to get into supervisor state, and are going to overreach the limits of the software design. Other corrective elements can be found in attempts to enhance the reliability and availability of applications.[133]

System audit trails

System surveillance, measurement and auditing are critical elements in providing the technical base for adequate security and integrity. The effectiveness and operability of the entire system, especially the protection mechanisms, must be continually scrutinized and measured. Management must be assured that the protection is in place and effective. Management must also be able to detect and to respond to events that constitute system security threats. Many of the same mechanisms used for performance measurement also can be used for monitoring of the protection mechanisms and the integrity of the entire system. A properly functioning audit mechanism should allow the specification of certain system events (such as OPEN, LOGON, etc) to trigger an audit trail.[32] The interfacing of system measurement and surveillance activity with the auditor is the subject of much activity and research.[119]

CONCLUSION

As of early 1976, systems are in use which provide a high degree of computer security and integrity, and may provide the basis for systems accreditation. The MULTICS project at MIT has led to commercial marketing of the system by Honeywell and a multi-level security enhancement by MITRE and the USAF.[112] The General Electric Mark III® Service has long been known for its good security. Other operating systems have been designed with security as an objective,[53,86,94,106,113] and the efforts of IBM and Honeywell have been previously mentioned. Current research directions are outlined in the paper by Saltzer[111] and should see commercial reality sometime in the next few years. Awareness of the risks is being fostered by numerous seminars and conferences. Large organizations, both commercial and government, are funding the position of systems security officer or computer security manager. An association, the Computer Security Institute, has been formed to provide information to, and give voice to the growing number of specialists in the field.

Current state of the art would seem to allow quite flexible and cost-effective security measures. But in practice, protection is generally not elaborate, flexible or impenetrable.[37] As a result, most safeguards are imposed "after the fact", through a mixture of managerial controls and physical security. This type of control is largely ineffective, due to inconsistencies, lack of proper redundancy or incompleteness. It appears that this will be the case, even after computer systems come provided with flexible and effective protection mechanisms.

In 1969, Lance Hoffman said that much research is needed to design security controls and to evaluate computer access control methods.[66] Nothing has changed to alter this. When designers and implementors agree on the needs, and the computer and software providers supply the secure methods to use their products, it is still up to the user to provide the proper environment, the procedures and the management climate to implement the principles of "least privilege," compartmentalization, redundancy in controls and personnel awareness that are the necessary first step in provision of security, privacy protection and system integrity. Only then, shall we realize the goals of simple, economic, functionally capable and modular protection mechanisms.[114]

In conclusion, it is important to realize that we are talking about a complex technology, with many interfaces.[124] Because of the great need, the next few years should see a continued broadening of interest, the forcing of computer security protection because of privacy legislation, awareness of the economic consequences of security deficiences, increased risk management efforts by computer system implementors and increasing government regulation of the data processing industry.

COMPUTER SECURITY BIBLIOGRAPHY

1. Abbott, Robert P. et al, *A Bibliography on Computer Operating Systems Security,* Lawrence Livermore Laboratory, University of California, The RISOS Project, Report UCRL-51555, April 1974. Technically oriented bibliography of key articles and papers relating to internal protection for computer systems.
2. Allen, Brandt, "Danger Ahead—Safeguard Your Computer," *Harvard Business Review,* November-December 1968. One of the early, well publicized articles on computer security. Helped achieve an awareness of physical security requirements.
3. Allen, Brandt, "Embezzler's Guide to the Computer," *Harvard Business Review,* July-August, 1975. The author talks directly to the would-be embezzler and cautions him not to be concerned about getting caught. The real big embezzlement schemes are still working and the perpetrators are not being caught. Shows how to defraud through manipulation of payroll, accounts payables, inventory, shipping documents and accounts receivable records that are maintained in computers.
4. AICPA, *Audits of Service-Center Produced Records,* American Institute of Certified Public Accountants, Auditing Standards Division, New York, 1974. This is a guide for Certified Public Accountants for their use in examining and reporting on the financial statements of clients whose records are produced by a computer service center or time-sharing firm. It is the basis for third-party audit reviews of such firms, and is oriented toward security and controls.
5. Anderson, James P., *Computer Security Technology Planning Study,* USAF Electronic Systems Division (MCIT), ESD-TR-73-51, October 1972. The "classic" study of computer security requirements in the later 1970's.
6. Anderson, James P., "Information Security in a Multi-User Computer Environment," In *Advances in Computers,* Vol. 12, 1972, Morris Rubinoff, Editor, Academic Press, New York. Stresses the technical problems and prospects for protecting data or information in multi-user computer environments. Shows how it is possible and profitable to penetrate systems for purposes of manipulation or unauthorized viewing.
7. Baran, Paul, *On Distributed Communications: IX. Security and Tamper-Free Considerations,* Rand RM 3765, August 1964. An unclassified discussion of cryptography: It anticipates the distributed data network exemplified by the ARPA net. It also anticipates the need for secure communications links.

8. Bates, William S., "Security of Computer-Based Information Systems," *Datamation*, May 1970, pp. 60-65. A survey article, written by a Marine captain.

9. Beardsley, Charles W., "Is Your Computer Insecure?" *IEEE Spectrum*, January 1972. A survey article that updates the earlier work of Lance Hoffman.

10. Berg, John L. (Editor), *Exploring Privacy and Data Security Costs—A Summary of a Workshop*, U. S. Dept. of Commerce, National Bureau of Standards, NBS Technical Note 876, August 1975. On 2/20/75 nine informed DP professionals were invited to NBS to discuss the costs of implementing privacy legislation. This is an edited summary of those discussions.

11. Bergart, J. F. and Marvin Denicoff and David K. Hsiao, *An Annotated and Cross-Referenced Bibliography on Computer Security and Access Control in Computer Systems*, Ohio State University, Computer and Information, Science Research Center Report OSU-CISRC-TR-72-12, November 1971.

12. Bigelow, Robert P., "Legal and Security Issues Posed by Computer Utilities," *Harvard Business Review*, Vol. 43, September-October 1967, pp. 150-161.

13. Bigelow, Robert P., "The Privacy Act of 1974," *The Practical Lawyer*, Vol. 21, #6, September 1, 1975. Covers the 1974 Privacy Act in some detail, and shows the effect on computerized personal information systems.

14. Bingham, H. W., *Security Techniques for EDP of Multi-Level Classified Information*, RADC-TR-65-415, December 1, 1965. A classic study that predates much of the efforts in authorization technology. This was a study of military intelligence security control techniques for the Burroughs D825. Hardware and software techniques currently under study are explained in great detail.

15. Bisbey, Richard L. II and Gerald J. Popek, "Encapsulation: An Approach to Operating System Security," *Proceedings, 1973 ACM Conference*, San Diego, California, also, AD-771-758, October 1973, ARPA. Encapsulation is the removal of access controls from the host computer, and the placing of a mini-computer-based access control system at the peripheral interfaces. This is a logical extension of the hypervisor, or virtual machine monitor. It is one solution to problems of security in fourth generation computer architecture.

16. Bjork, L. A. and C. T. Davies, Jr., Technical Report: *The Semantics of the Preservation and Recovery of Integrity in a Data System*, International Business Machines, Inc., Systems Development Division, TR-02.540, December 1972, also in *Proceedings, SHARE Conference*, November 1973. Discusses a unified view of data base recovery mechanisms, and proposes a set of concepts and principles revolving around spheres of control.

17. Branstad, Dennis K., "Security Aspects of Computer Networks," *Proceedings, AIAA Computer Network Conference*, Huntsville, Alabama, April 1973. Discusses the relevant issues of communications protocol, switching techniques and protection techniques in relation to design of secure computer/communications networks.

18. Branstad, Dennis K. and Susan K. Reed (editors), *Controlled Accessibility Workshop Report*, US Department of Commerce, National Bureau of Standards, NBS Technical Note 827, May 1974. Presents the results of a three day workshop in San Diego, sponsored by NBS and ACM. About 75 computer security professionals were invited, and discussed at length the technical and managerial issues of protection in computer systems. The results are significant, because much developmental work since then can be traced to the workshop.

19. Branstad, Dennis K., "Encryption Protection in Computer Data Communications," *Proceedings: Fourth Data Communications Conference*, Quebec City, Canada, October 1975, IEEE, New York. Encryption can be an effective process for protecting data during transmission. The degree of protection depends on the encryption algorithm, the implementation of the algorithm, and the associated administrative procedures. Additional security requirements of user identification, access authorization and auditing may be satisfied by combining encryption technology with a network access control machine (computer). This paper presents the proposed Federal encryption standard and the security requirements satisfied by proper use of the algorithm. It also discusses implementation.

20. Brown, William (editor), *Computer and Software Security*, AMR International, New York, 1971. This book contains the essential contents of the AMR Seminar on computer security. Physical security, development of a computer security plan, backup, legal, auditing, insurance, software and cryptographic techniques are all covered with varying levels of competency and detail. Included is a key-word-in-context bibliography covering citations up to mid-1971.

21. Browne, Peter S., "Computer Security—A Survey," *Data Base*, Quarterly Publication of ACM Special Interest Group on Business Data Processing, Fall 1972. An annotated bibliography of over 100 items presents the 'state of the art' as of 1972. It makes reference to some little publicized material, and shows the importance of then classified Department of Defense activity in the field.

22. Browne, Peter S. and Dennis D. Steinauer, *A Model for Access Control*, 1971 ACM/SIGFIDET Workshop on Data Description, Access and Control, San Diego, Nov. 11-12, 1971. Authorization is discussed from the standpoint of requirements for access to given objects. A conceptual model, based on the work of Weissman is developed.

23. Browne, Peter S., "Computer Security—A Risk Management Approach," *Security Register*, December 1973. Overviews the problems of security, shows how to systematically approach a computer security program, and calls for a risk management approach.

24. Browne, Peter S., *Security in Computer Networks*, NBS Special Publication 404, U. S. Dept. of Commerce, National Bureau of Standards, 1974. Also in *Data Communications User*, January 1975. Makes the point that security in networks is an extension of security in multi-user computer systems. Physical and administrative controls come first, followed by the technical requirements of hardware, software and encryption security controls.

25. Buskin, Arthur A., *A Framework for Computer Security*, SDC Technical Memorandum, TM-WD-5733, System Development Corporation, McLean, Virginia, June 1975. Presents an overview of the computer security problem and an interrelated set of axioms and principles of computer security as the beginning of a top-down, structured approach.

26. Canadian Institute of Chartered Accountants, *Computer Control Guidelines*, Canadian Institute of Chartered Accountants, 1970. An early attempt to define controls in computer systems for auditor guidelines.

27. Canning, Richard G., "Data Security in the CDB," *EDP Analyzer*, May 1970.

28. Canning, Richard G., "Security of the Computer Center," *EDP Analyzer*, December 1971.

29. Canning, Richard G., "Computer Security: Backup and Recovery Methods," *EDP Analyzer*, January 1972.

30. Canning, Richard G., "Protecting Valuable Data," *EDP Analyzer*, December 1973, Vol. 11, No. 12. *EDP Analyzer*, January 1974, Vol. 12, #1.

31. Canning, Richard G., "Computer Fraud and Embezzlement," *EDP Analyzer*, April 1975, Vol. 13, #4. These issues, written by one of the world's experts in business

data processing, are each a concise yet complete exposition of current thinking on each topic.

32. Chastain, Dennis R., "Security vs. Performance," *Datamation*, November 1973. Scrambling devices, file access validation and subversion tests are all part of a security environment. The author discusses his measurement efforts at the Defense Intelligence Agency regarding the overhead of these mechanisms.

33. Clements, Don and Lance J. Hoffman, *Computer Assisted System Design*, Electronics Research Laboratory, College of Engineering, Univ. of California, Berkeley, CA, ERL-M468, November 1974. Describes a computer software package that partially automates the selection of security techniques applicable to a particular system design.

34. Computer Security Research Group, *Computer Security Handbook*, Computer Security Research Group, Douglas B. Hoyt, Chairman, Macmillan and Company, Inc., New York, October 1973. Provides detailed information on management's role in the accountability and reporting, hardware/software controls, computer risk insurance, auditing computerized systems and outside contract services. The Computer Research Group was sponsored by the Atlantic and New Jersey Chapters of Association for Systems Management. Authors are Arthur Hutt, Belden Menkus, Eugene Redmond, Seymour Bosworth, Ralph Jones, Herbert Dickson, Robert Daley, Dick Brandon, Joseph Wasserman, Theodore Christiansen, Guy Migliaccio and Stephen Falb.

35. Conn, Richard W. and Richard H. Yamamoto, *A Model Highlighting the Security of Operating Systems*, RISOS Project, Lawrence Livermore Laboratory, Proceedings, ACM 1974 Conference, San Diego, CA. Penetration of computer systems have led several authors to identify generic weaknesses in operating systems, but have not led to formal methods of analysis. An approach showing promise in identifying trouble spots, as well as characterizing existing operating systems in a more general sense, lies in forming graph models in which nodes are program modules or data structures, and arcs are access or shared resource synchronization paths. A given system should be capable of reduction to a graph of this sort by appropriate analysis of its load modules.

36. Conway, R. W., "On the Implementation of Security Measures In Information Systems," *Communications of the ACM*, Vol. 15, No. 4, April 1972. Excellent presentation of those concepts that are germane to authorization and efficiency. All authorization does not have to be accomplished at run-time. To the extent that privacy is data independent, the access function is inexpensive to implement.

37. Conway, R. W., W. L. Maxwell, and H. L. Morgan, "Selective Security Capabilities in ASAP—A File Management System," *1972 Spring Joint Computer Conference*, p. 1181, May 1972. Shows implementation of principles described in the previous article.

38. Cotton, Ira W. and Paul Meissner, "Approaches to Controlling Personal Access to Computer Terminals," *Proceedings of the 1975 Symposium, Computer Networks, Trends and Applications*, National Bureau of Standards Institute for Computer Science and Technology. Considers a number of approaches to protection against unauthorized access to computers. Surveys the current state of the art of personal identification. Explains how devices can be compared, and introduces criteria that can be used in personal identification system evaluation and/or comparison.

39. Courtney, Robert H. Jr., *Forty Commonly Found Deficiencies in the Security of Data Processing Activities*, IBM, June 1971. A common sense primer to management, outlining frequently overlooked security deficiencies.

40. Courtney Robert H., Jr., *Security Risk Assessment in Electronic Data Processing Systems*. National Bureau of Standards, Task Group 15, October 1975. An approach toward determination of risk to data processing is postulated. It shows how to quantify the potential benefits afforded by given security protection for comparison with costs.

41. Davies, C. T., Jr., *A Recovery/Integrity Architecture for a Data System*, IBM, Systems Development Division, May 1972. Discusses concepts of integrity as related to operating system and data base architecture.

42. Dean, Albert L., Jr., "Data Privacy and Integrity Requirements for On-Line Data Management Systems," *1971 ACM-SIGFIDET Workshop on Data Description, Access and Control*, Nov. 1971, San Diego, CA. Identifies security requirements for an on-line data base management system. Concepts were implemented in work by the author for the US government.

43. Denning, Peter J. (Chairman), *An Undergraduate Course on Operating Systems Principles (Module 6-Protection)*, Interim Report of the COSINE Committee of the National Academy of Engineering (Commission on Education), June 1971. Embodies the principles of protection as discussed in the paper by Graham and Denning.

44. Department of Defense, *ADP Security Manual: Techniques and Procedures for Implementing, Deactivating, Testing and Evaluating Secure Resource-Sharing ADP Systems;* also see *Industrial Security Manual for Safeguarding Classified Information*, DOD 5200.28M, January 1973 and DOD 5220.22M, April 1970, order from Superintendent of Documents, US Government Printing Office, Washington, DC 20402. These two manuals provide very helpful guidance for non-DOD government and commercial organizations in defining and implementing physical and data processing security programs. Even though the concepts of the DOD classification of information hierarchy are pervasive, the methods, ideas and procedures are valuable in any environment.

45. Chadwick, H. A., "Burning Down the Data Center," *Datamation*, Vol. 21, #10, October 1975, pp. 60-64. A well written article that discusses DP insurance from the point of view of the data processing expert. Complex insurance related terms and concepts are clearly explained. Guidance is given as to what insurance coverage is needed, why various forms should be considered and who can provide insurance services to DP installers.

46. Enger, I. Sador, Guy T. Merriman and Ann L. Bussemy, *Automatic Security Classification Study*, RADC TR 67-472, October 1967. Report of an investigation of the feasibility of using computers to assign the government security classification to textual material. The "correctness" was only 54% but the techniques used did show promise for further research.

47. Federal Fire Council, *Recommended Practices No. 1—Fire Protection for Essential Electronic Equipment*, March 1962, Clearinghouse for Federal Scientific and Technical Information. Discusses practices in dealing with the threat of fire. Is the most thorough and concrete guidance to date.

48. Feistel, H., *Cryptographic Coding for Data Bank Privacy*, IBM Research Report, RC-2827, March 1970, also in *Scientific American* as "Cryptology and Computer Privacy," May 1973. Discusses concepts of cryptography that eventually led to the development of an IBM pilot project and the Federal encryption standard.

49. Fenwick, William A., "Marketing EDP Services: Reviewing the Legal Considerations," *Computers and Automation*, November 1971. A misnomer. The author is really talking about security measures to protect the confidentiality of data.

50. FIPS PUB-39 *Glossary of Terminology for Computer Systems Security*) Federal Information Processing Standards

Task Group 15: Computer Systems Security, National Bureau of Standards, US Department of Commerce, Washington, DC. September, 1975. A glossary of 70 terms relating solely to the concepts of privacy and computer security. The terms were exacted from many sources and refined through the joint efforts of FIPS Task Group 15, which was established in 1975 to develop standards and guidelines in computer systems security. Emphasis is on technical terms that relate to computer security architecture and communications security.

51. Foster, Caxton C., "Data Banks—A Position Paper," *Computers and Automation*, March 1971, p. 28. A summary of what can go wrong (machine failure, logical errors, eavesdropping, wiretapping) and what to do about it.

52. Friedman, T. D., "The Authorization Problem in Shared Files," *IBM Systems Journal* #4, 1970, pp. 258-280. The problem of sharing information yet providing proper authorization is reviewed. A model of a secured file system provides the basis for much current work in the field. Friedman rejects the hierarchial approach to authorization in favor of compartments or categorization of access rights.

53. Gaines, R. Stockton, An Operating System Based on the Concept of a Supervisory Computer," *Communications of the ACM*, Vol. 15, No. 3, March 1972. Concepts of protection are outlined.

54. Girsdansky, M. B., "Cryptology, the Computer, and Data Privacy," *Computers and Automation*, April 1972. Also see *Data Privacy: Cryptology and the Computer at IBM Research*, IBM Research Reports, Vol. 7, #4, 1971. An interesting study on what researchers are doing to devise 'unbreakable' codes and how many classical approaches to encipherment are easily compromised. The paper discusses "Lucifer," a hardware encryption device.

55. Glaser, E. L., "A Brief Description of Privacy Measures in the MULTICS Operating System," *SJCC 1967*, Vol. 30, pp. 303-304. Must reading for the student of protection theory.

56. GMIS, *An Administrative Guideline: Security and Confidentiality for Government Data Centers*, GMIS, December 1973. A report for state and local government members of GMIS that views computer security from an organizational viewpoint, and structures guidance on legislation, administrative control, personnel security, data flow security, physical security, hardware/software protection and confidentiality codes of ethics.

57. Goldstein, Robert C., "The Cost of Privacy, *Datamation*, Vol. 21, #10, October 1975, pp. 65-71, also see PhD dissertation, published by Honeywell Information Systems, Brighton, MA, 1975. Discusses implementation requirements for operators of personnel data systems in order to comply with privacy legislation. Physical security and clerical costs appear to be high-cost categories. Some possibilities for reducing the impact of privacy legislation are outlined.

58. Goode, George E. "Security for Teleprinters and Data Communications," *Data Management*, January 1973. Decribes cryptographic methods for securing communications.

59. Graham, G. Scott and Peter J. Denning, "Protection—Principles and Practice," *1972 Spring Joint Computer Conference*, p. 417, May 1972. Graham and Denning have analyzed most existing systems and find they fit the protection model that is described. The work draws heavily on theories of the "Princeton" school.

60. Graham, Robert M., "Protection in an Information Processing Utility," *CACM*, Vol. 11, #5, May 1968, p. 368. Describes the "rings of protection" in Multics.

61. Guide International, *Data Center Security Guidelines*, Guide Data Center Security Project—GSD 28-070, February 1972. Provides a set of guidelines for implementation and management of security in a data processing operations environment. Developed from a dedicated project at GUIDE, the major IBM user group.

62. GUIDE SHARE, *Data Base Management System Requirements*, Joint GUIDE-SHARE Data Base Requirements Group, November 11, 1970. An important document that outlines idealized requirements for data base management. Security and integrity play a dominant role.

63. Held, Gilbert, "Locking Intruders Out of a Network," *Data Communications*, January-February, 1975, pp. 27-31. The author favors a password scheme for controlling access to network components. Security is gained through use of nonprinting characters. Interesting statistics are presented on compromise possibilities utilizing a minicomputer and repeated attempts to exhaustively enumerate possible password combinations.

64. Healy, R. J., *Emergency and Disaster Planning*, New York, Wiley Press, 1969. An industrial security expert writes on the principles of emergency planning.

65. Hemphill, Charles F., Jr., *Security for Business and Industry*, Homewood, Illinois, Dow Jones, Irwin, Inc. 1971. This book has very little to do with EDP (only one chapter talks about computer room safeguards) but the principles of physical security are worth reading.

66. Hoffman, Lance J., "Computers and Privacy: A Survey," *Computing Surveys*, Vol. 1, #2, June 1969. The "classic" survey on this subject.

67. Hoffman, Lance J., "The Formulary Model for Flexible Privacy and Access Controls," *FJCC*, November 1971, Las Vegas, Nevada. Presents a model for interfacing user access controls and data by means of coded procedure called formularies.

68. Hoffman, Lance J. and W. F. Miller, "Getting a Personal Dossier From a Statistical Data Bank," *Datamation*, May 1970. An interesting example of how to input information by indirect means from innocent files.

69. Hoffman, Lance J. (editor), *Security and Privacy in Information Systems*, Melville Publishing Company, Los Angeles, California, 1973. The book developed from a collection of readings used in a graduate course on the technological methods of providing security in computer systems.

70. Hollingsworth, Dennis, Steve Glaseman and Martha Hopwood, *Security Test and Evaluation Tools: An Approach to Operating System Security Analysis*, Rand Corporation, p. 5298, September 1974. As of this paper, the techniques for determining the security characteristics of system software are primitive, based generally upon the notion of penetration testing, and manual examination of system source code. The paper suggests ways of developing and refining the tools of operating system security analysis.

71. Honeywell Information Systems, "Computer Security and Privacy," *Symposium Proceedings*, April 1975. A symposium sponsored by Honeywell Information Systems. Includes 20 papers covering security approaches, requirements, technical solutions, and data center management.

72. Hsiao, David K., *A File System for a Problem Solving Facility*, PhD dissertation, University of Pennsylvania, 1968, published by NTIS, Springfield, VA, AD-671826. Hsiao discusses the use of an "authority item" which allows protection below the file level. The system is based on a multilist file structure.

73. Hunt, Kathleen and Rein Turn, *Privacy and Security in Databank Systems: An Annotated Bibliography, 1970-1973*, Rand Corporation, Santa Monica, CA, R-1351-NSF, March 1973.

74. International Business Machines, *Data Security and Data Processing*, (Volumes 1-6) IBM, Data Processing Division, June 1974. Presents the findings of the May 1972 data study project at MIT, TRW Systems, State of Illinois and IBM's Federal Systems Division. The IBM Resource

Security System (RSS) was evaluated. Results of cost studies and implementation measurements as well as general papers on the subject are included in this unevenly presented, but valuable collection.

75. International Business Machines, Inc., *The Considerations of Data Security in a Computer Environment*, IBM, Data Processing Division, 1968. A widely distributed monograph on data security: it helped bring attention to some of the needs and the problems.

76. International Business Machines, Inc., *The Considerations of Physical Security in a Computer Environment*, IBM, Data Processing Division, 1972. A companion to the data security monograph of 1968; it draws largely upon the work and experiences of Robert Courtney.

77. International Business Machines, Inc., *The Fire and After the Fire*, IBM, Data Processing Division, G520-2741-0, January 1973. A marketing oriented brochure that explains how IBM averted a major disaster by its implementation of a backup and recovery plan following a fire at their Program Information Department facility.

78. International Business Machines, Inc., *Proceedings: IBM Data Security Forum*, Denver, Co., September 1974. IBM, Data Processing Division, G520-2965-0, 1974. Contains a number of papers on data security related topics, including risk management, hardware protection, and operational controls to enhance data security.

79. Jacobson, Robert V., Peter S. Browne and William F. Brown (editors), *Guidelines for Automatic Data Processing Physical Security and Risk Management*, US Department of Commerce, National Bureau of Standards, FIPS PUB 31, June 1974. Basic reference guidelines for implementation of physical security and risk management. Reference is made to numerous sources of information that will aid an installation manager in defining security requirements and making essential security decisions.

80. Jacobson, Robert V., "Cornerstones for Computer Security," *Security Register*, Vol. 1, No. 2, January-February, 1974. Discusses some of the fallacies of the "amulet" approach towards computer security, as traditionally practiced. Risk analysis, quality control, contingency planning and independent audit are postulated as the four cornerstones of computer security.

81. Kahn, David, *The Code Breakers*, The MacMillan Company, New York, 1967. The classic book for those interested in cryptology and cryptanalysis.

82. Karush, A. D. and Larson, R. H., *Analysis and Measurement of the Audit Recording Function*, System Development Corp., TM-4435, August 1969. Early research in system audit mechanisms.

83. Krauss, Leonard I., *Administering and Controlling the Company Data Processing Function*, Prentice Hall, Englewood Cliffs, NJ, 1969.

84. Krauss, Leonard I., *SAFE—Audit and Field Evaluation for Computer Facilities and Information Systems*, Firebrand, Krauss and Co. Inc., 1972. An extensive checklist audit guide for a complete review of security and controls in data processing facilities. Uses a weighted scoring that attempts to quantitatively score the merit of controls noted.

85. Lampson, Butler W., "Dynamic Protection Structures," *Proceedings, 1969 Fall Joint Computer Conference*, pp. 27-38. AFIPS Press, Montvale, NJ. Lampson's theory of protection is a foundation for much modern day research and application of protection in operating systems and computer hardware.

86. Lipner, Steven, "A Minicomputer Security Control System," *Compcon 74*, San Francisco, February 1974. Lipner is a pioneer in espousing and developing a protected hardware/software system based on the concept of a "security kernel," a certifiably small, protected module that itself is the authorization mechanism for other system components.

87. Martin, James, *Security, Accuracy and Privacy in Computer Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1973. To date, the most complete treatment on the subject of computer security.

88. Martin, James and Adrian Norman, *The Computerized Society*, Prentice-Hall, Englewood Cliffs, NJ, 1970, pp. 481-498. Security and privacy are extensively treated in this book which explores the effect of computers on society.

89. Menkus, Belden, "Computer Security Needs a Common Sense Approach," *Administrative Management*, March 1973. Discusses two aspects of comprehensive physical security. The first step is to build security into the facility by making it inconspicuous, installing access controls, and providing basic environmental support. The second step is to ensure integrity of processing through controls over input and file access, and ensuring good facility operating procedures.

90. MITRE Corporation, *The Privacy Mandate—Planning for Action*, National Bureau of Standards and MITRE Corp. Washington DC, August 1975. A summary of a workshop sponsored by the publishing organizations to develop recommendations for action in implementing privacy legislation. Four working panels covered the issues of individual privacy rights, institutional responsibilities, technological implications and the economics of privacy. Viewpoints of many interested organizations are also included. Unfortunately, the proceedings do not capture the depth of discussion that actually took place.

91. Molho, L. M., *Hardware Aspects of Secure Computing*, System Development Corporation, SP3453, December 1969 and *Proceedings, 1970 SJCC*, AFIPS Press, Montvale NJ.

92. National Bureau of Standards, *Computer Security Guidelines for Implementing the Privacy Act of 1974*, Department of Commerce, National Bureau of Standards, FIPS PUB 41, September 1975. Provides a set of guidelines for the use of technical procedures for safeguarding personal data in automated information systems. Covers physical security procedures, information management practices and computer system/network security controls.

93. National Fire Protection Association, *Standards for the Protection of Electronic Computer Systems*, NFPA Standard 75, May 1962. Covers fire detection and suppression equipment requirements.

94. Neumann, Peter, "On the Design and Verification of a Secure Operating System," *Proceedings, 1974 National Computer Conference*, AFIPS Press, pp. 978-979. Neumann of Stanford Research Institute, has been doing work for government agencies and others in proving the correctness of software, with ultimate security and protection implications.

95. Noll, A. Michael, "The Interactions of Computers and Privacy," *Honeywell Computer Journal*, Vol. 7, #3, 1973. A survey of the existing relationship between computer usage and the concepts of confidentiality, security and privacy. Explores where new problems raised by technology show gaps and inadequacies in laws. Covers computer security threats, levels, costs and surveys the technological aspects of computer security.

96. Notz, W. A. and J. L. Smith, *An Experimental Application of Cryptography to a Remotely Accessed Data System*, IBM Corp., RC 3508, August, 1971. Describes "Lucifer," a hardware encryption and decoding device attached to a time-shared IBM 360/67.

97. Office of Emergency Preparedness, *Disaster Preparedness*, Report to the Congress by Executive Office of the President, Office of Emergency Preparedness, US Government Printing Office, January 1972. A comprehensive study

useful for the data processing risk manager. Discusses and quantifies risks due to floods, wind, fire, earthquakes, landslides, volcanos, freezes and droughts.

98. Owens, Richard C. Jr., "Evaluation of Access Authorization Characteristics of Derived Data Sets," *1971 SIGFIDET Workshop on Data Definition, Access and Control*, pp. 263-278, ACM, New York. Project MAC—TR89, July 1971, also NTIS AD-728036. The two papers by Owens describe the access control for the MACAIMS Data Management project at MIT. The concepts are quite sophisticated.

99. Palme, Jacob, "Software Security," *Datamation*, January 1974, Vol. 20, #1. Discusses the prevention of illegal access to, modification of and interference with data. A general, survey type article.

100. Parker, Donn B., Susan Nycum and S. Stephen Gura, "Computer Abuse," *Stanford Research Institute, Final Report*, NSF Grant GI-37226, Report NSF/RA/S-73-017, November 1973. Describes results of a study of 148 cases of computer abuse. Provides technical, legal, and social perspectives on computer crime. The purpose of the report is to alert business and government users of the seriousness, extent, and potentials of computer abuse as a new and emerging social and technological problem.

101. Parker, Donn B. and Susan Nycum, "The New Criminal," *Datamation*, January 1974, Vol. 20, #1. Discusses the old Trojan Horse program trick, and others that have netted millions for enterprising, crooked data processing personnel.

102. Patrick, Robert B. (editor), *AFIPS System Review Manual on Security*, AFIPS Press, Montvale, NJ, 1974. The results of a two year study under the direction of a committee chaired by John Gosden of Equitable Life Insurance Co. It consists of a set of guides for evaluation and a series of checklists to aid in the review of system security.

103. Peters, Bernard, "Security Considerations in a Multiprogrammed Computer System," *Proceedings, Spring Joint Computer Conference*, Vol. 30, 1967, pp. 283-286. One of the pioneering articles dealing with the issues of computer security. Based on government agency work in the mid-1960's.

104. Peterson, H. E. and Turn, Rein, "System Implications of Information Privacy," *Proceedings, Spring Joint Computer Conference*, Vol. 30, 1967, pp. 291-300. Another pioneering work based on the technical implications of privacy.

105. Pfaff, Alfred M., *Toward A Taxonomy of Computer Security Requirements for Federal Agencies*, Federal Information Processing Standards Task Group 15: Computer System Security, National Bureau of Standards, US Department of Commerce, Washington, DC, September, 1975. Computer security is defined as organized into the three distinct aspects of processing integrity, data integrity and data confidentiality. Security requirements are mapped to particular security countermeasures, and a system for rating the degree of compliance is proposed. A very useful addition is the extraction of relevant portions of US Codes (Public Law) relating to computer security.

106. Popek, Gerald J. and C. Kline, "Verifiable Secure Operating System Software," *Proceedings, 1974 National Computer Conference*. AFIPS Press, pp. 145-152. Popek's work is very significant in that it provides much of the basis for current research and experimentation in secure operating systems.

107. Reed, Susan K. and Martha M. Gray, *Controlled Accessibility Bibliography*, US Department of Commerce, National Bureau of Standards, NBS Technical Note 780, June 1973. A Comprehensive, technically oriented bibliography prepared in conjunction with the San Diego Controlled Accessibility Workshop.

108. Reed, Irving S., *The Application of Information Theory to Privacy In Data Banks*, Rand Corporation (NSF), R-1282-NSF, May 1973. Covers theoretical, mathematical aspects of information protection.

109. Renninger, Clark R. and Dennis K. Branstad (editors), *Government Looks at Privacy and Security in Computer Systems*, US Department of Commerce, National Bureau of Standards, NBS Technical Note 809, February, 1974. Potential confrontations between society and technology over problems of individual privacy and data confidentiality can be defused by understanding and action. A conference on privacy and security was held at NBS, November 19 and 20, 1973. A number of speakers provided statements of governmental needs and problems. Also suggested was a broad range of activities for satisfying the needs.

110. Renninger, Clark R. (Editor), *Approaches to Privacy and Security in Computer Systems*, US Department of Commerce, National Bureau of Standards, NBS Special Publication 404, September, 1974. This publication summarizes and contains the proceedings of a conference held at NBS on March 4-5, 1974 to continue the dialog in search of ways to protect confidential information in computer systems. Proposals were presented for meeting governmental needs for safeguarding data confidentiality. Among the proposals were the enactment of privacy legislation, improved computer system architecture and access controls, information and security management guidelines and the development of systematic, balanced approaches to system security. A number of prominent computer, legal and social professionals presented their views as to potential solutions.

111. Saltzer, Jerome H., "Ongoing Research and Development on Information Protection," *Operating Systems Review*, July 1974; also *Proceedings, Computer Security and Privacy Symposium*, Honeywell Information Systems, April 1975. A survey of current research in the technical solutions to computer security.

112. Saltzer, Jerome H., "Protection and the Control of Information Sharing in MULTICS," *Communications of the ACM*, Vol. 17, #7, July 1974. Describes the protection mechanisms in the MULTICS system. These are some of the most advanced in current implementation.

113. Saltzer, Jerome H. and Michael D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, IEEE Computer Society, September 1975. A thorough discussion of the technical aspects of providing protection in computer systems. This is the most complete and most valuable discussion of the concepts of protection to date.

114. Schroeder, Michael D. and Jerome H. Saltzer, "A Hardware Architecture for Implementing Protection Rings," *Proceedings, 3rd Annual ACM Symposium on Operating Systems Principles*, October 1971. Schroeder and Saltzer have designed hardware for use on the MULTICS system. It has found its eventual implementation on the Honeywell 6180. It allows efficient and flexible access authorization to be implemented partially in hardware.

115. Shannon, C. E., *Communications Theory of Secrecy Systems*, Bell Telephone System Technical Journal, October 1949, Vol. 28, #4, pp. 656-715. The theory of cryptology has not been significantly improved since this landmark, unclassified study was published.

116. Sorensen, J. L., "Common sense in computer Security," *Journal of Systems Management*, April 1972, pp. 12-14. A hypothesis is made that computer security is nothing more than rational decision making.

117. Stern, Ludwig, "Contingency Planning: Why? How? and How Much?," *Datamation*, September 1974, Vol. 20, #9, pp. 83-95. Discusses an approach to contingency planning that has been implemented at a major corporation.

118. Turn, Rein, *Privacy and Security in Personal Information*,

The Rand Corporation, Santa Monica, CA, R-1044-NSF, March 1974. This report presents the results of a National Science Foundation research study on theoretical and technical aspects of protection of personal information in databanks. The protection requirements and design of protection is the key focus. The investigation led to the establishment of classifications of systems and the sensitivity of personal information and the development of a protector-intruder model.

119. Turn, Rein, *Remarks on the Instrumentation of Databank Systems For Data Security*, The Rand Corporation, Santa Monica, CA, P-5151, January, 1974. This paper discusses the information requirements of an active security subsystem as well as auditing and threat monitoring. It explores ways of instrumenting a databank system for obtaining this information.

120. Turn, Rein and Norman Z. Shapiro, *Privacy and Security in Databank Systems: Measures of Effectiveness, Costs and Protector-Intruder Interactions*, The Rand Corporation, Santa Monica, CA, P-4871, July 1972, also in *Proceedings, Fall Joint Computer Conference*, 1972, AFIPS Press, Montvale, NJ. Introduces a model that attempts to systematicize the process of measuring the "malicious" penetrator of computer systems.

121. Van Tassel, Dennis, *Computer Security Management*, Prentice-Hall, Inc., Englewood Cliffs, NJ, April 1972. One of the first books to appear concerning computer security. It is largely a collection of previous articles by the author and a series of "horror stories."

122. Walter, K. G. et al, *Modeling the Security Interface*, Department of Computing and Information Science, Case Western Reserve University, Cleveland, Ohio, Report #1158, August 1974. Presents developments in modeling a security kernel.

123. Ware, Willis H., *Computer Data Banks and Security Controls*, The Rand Corporation, Santa Monica, CA, P-4329, 1970. A pioneering monograph by the dean of computer security professionals.

124. Ware, Willis H., "Security and Privacy: Similarities and Differences," *Proceedings, Spring Joint Computer Conference*, Vol. 30, 1967, pp. 287-290. Along with the early work by Brandt Allen and Joseph Wasserman, Dr. Ware helped bring awareness of computer security to the attention of management and data processing professionals prior to 1970.

125. Wasserman, Joseph J., "Plugging the Leaks in Computer Security," *Harvard Business Review*, September 1969, pp. 119-129. One of the early and most thorough efforts to provide a framework for operational controls in computer systems.

126. Wasserman, Joseph J., "Selecting a Computer Audit Package," *Journal of Accountancy*, April 1974. Explains a methodology and approach toward audit package evaluation.

127. Weiss, Harold, "Computer Security: An Overview," *Datamation*, Vol. 20, #1, January 1974. Even though computer crime is increasing, fire, earthquakes and storms are postulated as the greater hazard to computer systems. Few installations have taken even the simple steps toward protection.

128. Weissman, Clark, "Security Controls in the ADEPT-50 Timesharing System," *Proceedings, Fall Joint Computer Conference*, Vol. 35, 1969, pp. 119-133. Describes the mechanisms used in implementing security in a large, data management oriented Department of Defense computer system.

129. Weissman, Clark, *System Security Analysis/Certification Methodology and Results*, System Development Corporation, Santa Monica, CA, SP-3728, October 1973. Presents an approach toward system certification.

130. Weissman, Clark, *Tradeoffs in Security System Design*, System Development Corporation, Santa Monica, CA, SP-3548, September 1970, also in *Data Management*, April 1972. A very important paper that clearly presents the issues of trade-off analysis in designing and implementing protection.

131. Wessler, John, Edith Myers and W. David Gardner, "Physical Security—Facts and Fancies," *Datamation*, July 1, 1971. Another survey article on physical security.

132. Westin, Alan F., *Privacy and Freedom*, Atheneum Press, New York, NY, 1967. The earliest and one of the clearest books on the subject.

133. Winkler, Stanley and Lee Danner, "Data Security in the Computer Communication Environment," *Computer*, Volume 7, No. 23, February 1974, IEEE. Describes security concerns in multi-terminal computer systems. Useful as an introduction to the problems and the nature of network security. Describes a number of possible implementations of controlled access to data.

134. Yourdan, Edward, "Reliability of Real-Time Systems," *Modern Data*, January-June 1972. A six-part series that thoroughly explores data integrity and reliability. Covered are the different concepts of reliability, causes of system failure, examples of failure and approaches to error recovery.

Computer abuse perpetrators and vulnerabilities of computer systems

by DONN B. PARKER
Stanford Research Institute
Menlo Park, California

ABSTRACT

Analysis of computer abuse experience is valuable in threat and risk studies performed to develop appropriate safeguards in computer use. A profile of computer abuse perpetrators has been developed on the basis of interviews with 17 offenders involved in a total of 15 cases. Common characteristics, occupations, and modus operandi are documented and analyzed. Computer systems' and user organizations' vulnerabilities that facilitated perpetrators' actions are also described, based on study of 375 reported cases of abuse. Eight main vulnerable functions and nine main vulnerable functional locations are identified and ranked by incidence of occurrence. Each vulnerability is described by examples in the form of brief case descriptions. Finally, priorities for safeguards are deduced from the results of the study.

INTRODUCTION

Computer abuse research has been conducted over the past five years at Stanford Research Institute, supported in part by the National Science Foundation (Grants GI-37226 and GJ-44313). Computer abuse is defined as any intentional act in which one or more victims suffered or could have suffered a loss, and one or more perpetrators made or could have made a gain.

The assessment of computer abuse and development of a case file—which now contains information on about 375 cases—are now sufficiently advanced to allow analyses that can assist security planners and EDP management. Two basic areas of concern are the sources of threats—the computer abuse perpetrators—and the vulnerabilities that facilitated their acts. With the rubric, "know the enemy to overcome him" in mind, a profile of known perpetrators was developed and documented. The admonition to be aware of the vulnerabilities of victims is the motivation for also identifying and presenting the weaknesses and functional locations of weaknesses among the known, reported cases of computer abuse.

COMPUTER ABUSE PERPETRATORS

An important aim of computer abuse research is determining a typology of known perpetrators as an aid in developing safeguards. Such a typology can be used in reducing the number of possible perpetrators and their potential for doing harm.

Interviews of varying lengths were conducted with seventeen perpetrators. In some cases over 20 hours of interviews were held, involving numerous sessions covering pretrial, criminal trial, presentencing, incarceration, and post incarceration periods. In six cases only brief telephone conversations were held with perpetrators, but information from them was heavily supplemented with facts and opinions of other case participants. No attempt was made to carry out psychological profiling, but obvious characteristics were determined in a gross fashion by interviewers with expertise in computer technology, management, and law. The sample of perpetrators was chosen on the basis of geographic and interview schedule expediency, case notoriety, and technical novelty or frequency of the abuse method. In the future attempts will be made to choose perpetrators so that the growing sample will be more representative of the total file of cases.

Characteristics were collected and synthesized by interviewing perpetrators, and conclusions were based on computer technology management experience of the interviewers. Characteristics of white collar criminals were identified from criminology literature.[1-3] Theories and information from this source include the trust position vulnerability, Robin Hood, and differential association theories; known characteristics of people in EDP occupations such as their ages, skills, occupation-related actions, technical challenge and game playing interests; and characteristics discovered in interviews, such as tendencies to collusion and business and occupational aggressiveness. The characteristics identified are those that a manager of a computer system might recognize among people within the computer environment operating or affected by computer services.

There are ten characteristics of the typology and supporting data based on the sample of 17 perpetrators.

(1) Age—Perpetrators are young. Mean age is 29 years, median age is 25 years, and the range is 18 to 46 years.

(2) Skill Level—Skill level pertaining to the abuse is high, with professional and managerial skills predominant:

Skill Level/Occupation	*Number of Perpetrators*
Low experience technician	1
Technician	3
Low experience professional	1
High experience professional	5
Manager	7

(3) Relation Between Occupation and Abuse—In all cases except one, perpetrators performed their acts while engaged in their occupations. The exception is an individual who, while president of an electronic supply house, posed as a telephone company employee to order the delivery of telephone equipment. Eleven of the perpetrators violated their occupational positions of trust. Six performed their acts without violating occupational trust.

The perpetrators' occupations, associated with types and characteristics of the victims appear in Table I.

A significant range of types of perpetrators and victims is included with consumers, non-EDP employees, EDP employees, managers and staff, and large and small victims in various industries.

(4) Abuse Modi Operandi—The modi operandi were almost equally divided between unauthorized data manipulation during authorized computer use and unauthorized computer use. In eleven cases computers were primarily objects of acts; in five of the cases they provided the environment for the act; and in one case a computer was the instrument of the act. Eight cases involved batch-operated computer systems, five involved time-sharing systems, and four involved transaction systems.

Table II presents a range of perpetrators' technical acts.

Types of losses were: information or property fraud or theft in five cases, financial fraud or theft in ten cases, and unauthorized use of services in two cases. In the last two cases, perpetrators were able to use time-sharing services without paying for them and also took proprietary data, but no loss was sustained by victims of the data theft.

(5) Collusion—Collusion occurred in seven cases. Four of the cases involved only two people; the others involved five, seven and twenty-two people. Collusion was found necessary by the perpetrators either because they did not possess all of the skills, time, or resources necessary for the act, or they needed assistance in converting the act to financial gain. Several of the other perpetrators said they considered obtaining assistance from others but rejected the idea because they felt they should not entice others into wrongdoing.

(6) Personal Gain—For a group of eight of the cases, a total financial gain of $4 million was discov-

TABLE I—Computer Abuse Perpetrators' Occupations and Types of Victims

Perpetrator Occupations	Victims	Number of Perpetrators
Retail consumer	large insurance company	1
Teller	large bank	1
Accountant and computer service company owner	small manufacturing company (one accountant)	1
Time-sharing user	large service, small service, large private system	3
Business programmer	small banks	2
Systems programmer	state agency	1
Data input supervisor	large insurance company	1
Computer operations and systems managers	small bank, large insurance companies	3
Firm presidents	small electronic supply and software house	2
Business manager	large manufacturer	1
Sales manager	large time-sharing service	1

TABLE II—Perpetrators' Technical Methods of Computer Abuse

Number of Cases	Methods
2	Manipulation of data by unauthorized computer program changes.
1	Unauthorized use of an existing program to manipulate data.
3	Compromise or penetration of a time-sharing computer system by legitimate access from an on-line terminal performed by discovering and exploiting a weakness or error in the system controls that protect users or the system.
2	Impersonating an authorized terminal user of a time-sharing computer service by using confidential identification codes to obtain and use proprietary programs and data.
1	Use of a computer as an instrument or tool to plan or control a noncomputer related act.
2	Taking by manual means and/or selling copies of proprietary computer programs without the owner's permission or knowledge.
4	Inputting incorrect data and/or using incorrect output by authorized and correct means but for unauthorized purposes.

ered, with an average gain of $500,000 per case. The range for the 15 cases is $1400 to $1.5 million. Another type of gain was business or employment advantage over competitors through sabotage or espionage (intelligence gathering).

(7) Differential Association—Thirteen perpetrators demonstrated the differential association syndrome: The white collar criminal in his act deviates from accepted practices of his associates only in small ways.

(8) Robin Hood Syndrome—Twelve perpetrators exhibited the Robin Hood syndrome: They differentiate strongly between harming people, which is highly immoral within their standards, and harming organizations, which they can easily rationalize.

(9) Game Playing—Fifteen perpetrators indicated that they considered their acts games pitting their skills against the computer and the victim organization. The games represented challenges to them, and made their lives exciting and filled with danger. Fourteen perpetrators accepted the challenge with considerable aggressive behavior, identified by one perpetrator as the desire to participate in physically dangerous activities such as entering a bull ring or driving a race car.

(10) The dispositions of the perpetrators at this writing follow:

Disposition	*Number of Perpetrators*
Felony conviction	9
Felony charged	1
Lawsuit judgment	1
Charges dropped	1
No sanctions proposed	4
Hired by victim	1

Only one perpetrator, a nineteen-year-old programmer, had a prior conviction—for a misdemeanor of marijuana possession.

(11) Personal Characteristics—Generally the perpetrators were accepted as reliable, honest, bright, highly motivated in their work and most desirable people for a manager to hire. They do not appear special as a class and could not be classed as professional criminals who take pride in their wrongdoing. The greatest fear they reported occurring during their acts was unanticipated detection and exposure of their acts to their families, friends, and coworkers. This was feared more than incarceration. In fact, after sentencing, several said imprisonment was the best solution to the original problem that drove them to their acts. After they were caught, their greatest concern was to minimize the criminality aspects of their cases.

This initial study of perpetrators is enough to suggest the value of a thorough sociological and psychological study as a basis for identifying populations of potential perpetrators in automated crime.

COMPUTER SYSTEM VULNERABILITIES THAT FACILITATE ABUSE

Vulnerabilities to computer abuse must be understood for effective threat and risk analysis and computer security. Many vulnerabilities seem obvious, but the security planner can never be sure he has thought of them all or even the important ones. Two analyses, based on the principal vulnerability found or surmised in each of the 375 recorded cases of computer abuse were made to assist in this activity. The first was based on a breakdown of common functional weaknesses, such as inadequate input/output controls; the second was based on a breakdown of the most common functional and physical locations of vulnerabilities. Tables III(a) and III(b) summarize these vulnerabilities and locations.

FUNCTIONAL VULNERABILITIES

Eight primary functional vulnerabilities emerged from the analysis. They are listed below in order of frequency of occurrence. Each vulnerability is general enough to maintain an acceptable level of confidence in assignment of cases to types of vulnerabilities. This approach was adopted because the amount of information about some cases is limited. Examples from the file that demonstrate the range of acts facilitated by each vulnerability appear in the appendix.

(1) Poor Controls Over Manual Handling of Input/Output Data—This vulnerability was associated with 147 cases. The greatest vulnerability occurs wherever assets are most exposed. Over the past 17 years—the period of reported cases—assets have been most tangible and subject to human acts before entry into computers and after output from computers. Data assets are more accessible outside computers than when they are within them, and programs must be executed to achieve unauthorized access. Controls that are often absent or weak include separation of data handling and conversion tasks, dual control of tasks, document counts, batch total checking, audit trials, protective storage, access restrictions, and labeling.

(2) Weak or Nonexistent Physical Access Controls—This vulnerability to access to computing facilities accounted for 46 cases. Where physical access is the primary vulnerability, nonemployees have gained access to computer facilities, and employees have gained access at unauthorized times and in areas in which they were unauthorized. Perpetrators' motivations have included political, competi-

tive, and financial gain. Financial gain occurred mostly through unauthorized selling of computer services, holding computer centers for extortion purposes, burglary, and larceny. In a number of cases employee disgruntlement has been the motivating factor. In some of these cases disgruntlement stemmed from frustration with various aspects of automated society. Controls that were found to be weak or nonexistent include door access, intrusion alarms, low visibility of assets, identification and establishment of secure perimeters, badge systems, guard and automated monitoring functions (closed circuit television), inspection of transported equipment and supplies, and staff sensitivity to intrusion. A number of the intrusions occurred during nonworking hours when safeguards and staff who might notice intrusions were not present.

Four cases from the case file in which abuse was facilitated by physical access vulnerability involved attacks on computers with firearms; one involved a dispute over national politics; another case was perpetrated by a computer operator frustrated with his job, and the remaining two are presumed to have involved citizens frustrated in dealing with government bureaucracy and computer-based services.

(3) Computer and Terminal Operations Procedures—This vulnerability accounted for 43 cases. Losses resulting from operational procedures weaknesses have resulted from sabotage, espionage, sale of services and data extracted from computer systems, unauthorized use of facilities for personal advantage, and direct financial gain associated with negotiable instruments in operational EDP areas. The controls whose weakness or absence facilitates these kinds of acts include separation of operational staff tasks, dual control over sensitive functions, staff accountability, accounting of resources and services, threat monitoring, close supervision of operating staff, sensitivity briefings of staff, documentation of operational procedures, backup capabilities and resources, and recovery and contingency plans. The most common abuse problem has been the unauthorized use or sale of services and data. The next most common problem is sabotage perpetrated by disgruntled EDP operations staff.

(4) Weaknesses in Business Ethics—Abuse facilitated by this vulnerability accounted for 41 cases. A weakness or breakdown in business ethics can result in computer abuse perpetrated in the name of a business or government organization. The principal act is more related to a company's practices or management decisions rather than to identifiable unauthorized acts of individuals using computers. These practices and decisions result in deception, intimidation, unauthorized use of services or products, financial fraud, espionage, and sabotage in competitive situations. Controls include review of business practices by company boards of directors or other top level management, certified public accountant audits, and effective practices of regulatory and law enforcement agencies.

(5) Weaknesses in the Control of Computer Programs—This vulnerability facilitated 33 cases. Programs are assets subject to abuse. They can also be used as tools in the perpetration of abuse, and are subject to unauthorized changes to perpetrate abusive acts. The latter abuses are the most common. Controls found lacking include labeling programs to identify ownership, formal development methods (including testing and quality assurance), separation of programming responsibilities in large program developments, dual control over sensitive parts of programs, accountability of programmers for the programs they produce, the safe storage of programs and documentation, audit comparisons of operational programs with master copies, formal update and maintenance

TABLE III—Vulnerabilities to Computer Abuse

(Incidence in Reported Cases)

(a) Vulnerable Functions

Function	Number of Cases	Percent of Cases
Manual handling of input/output data	147	41%
Physical access to EDP facilities	46	13
Operations procedures	43	12
Business practices	41	11
Computer programs usage	33	9
Operating systems access and integrity	24	6
Time-sharing service usage	19	5
Magnetic tape storage	9	3
Totals	362*	100%

(b) Vulnerable Locations

			Total	
Functional Locations	Number of Cases	Percent of Cases	Number of Cases	Percent of Cases
Data and report preparation	120	33		
Terminal areas	14	4	134	37
Computer operations	95	26		
Terminal areas	10	3	105	29
Non-EDP	44	13	44	13
Computer systems	7	2		
Terminal systems	33	9	40	11
Programming	27	7	27	7
Magnetic tape storage	12	3	12	3
			362*	100%

* 13 of 375 cases were not amenable to analysis.

procedures, and establishment of ethical concepts of program ownership.

(6) Operating System Access and Integrity Weaknesses—This vulnerability facilitated 24 cases. All of these compromises of computer operating systems that are recorded involve the use of time-sharing services. Compromises are accomplished through discoveries of weaknesses in design or taking advantage of bugs or shortcuts introduced by programmers in the implementation of operating systems. The acts involve intentional searches for weaknesses in operating systems, or the unauthorized exploitation of weaknesses discovered accidentally. Most of the acts have been perpetrated in university-run time-sharing services by students committing vandalism or malicious mischief, or attempting to obtain computer time without charge. Controls that would eliminate weaknesses in operating systems include methods for proving the integrity and security of the design of operating systems, imposing sufficient implementation methods and discipline, proving the integrity of implemented systems relative to complete and consistent specifications, and adopting rigorous maintenance procedures.

(7) Poor Controls Over Access Through Impersonation to Time-Sharing Services—This vulnerability facilitated 19 cases. Unauthorized access through impersonation to time-sharing services can most easily be gained by obtaining secret passwords which are keys for the most common method of protecting users of time-sharing services. Perpetrators learn passwords that are exposed accidentally through carelessness or administrative failures, or obtain them by conning people into revealing their passwords or by guessing obvious combinations of characters and digits. It is suspected that this type of abuse is so common that few victims bother to report cases in recordable form. Control failures include poor administration of passwords, failure to change passwords periodically, failure of users to protect their passwords, poor choices of passwords, absence of threat monitoring or password-use analysis in time-sharing systems, and failure to suppress or obliterate the printing of passwords.

(8) Weaknesses in Magnetic Tape Control—This vulnerability accounts for nine cases. Theft of magnetic tapes, their destruction, and data erasure from them are acts attributed to weaknesses in control of magnetic tapes. Many other cases, identified as operational procedure problems, involved the manipulation of data on tapes and copying. (No cases are known in which magnetic disk packs have been subject to abusive acts.) Controls found lacking include limited access to tape libraries, safe storage of magnetic tapes, the labeling of tape reels, location and reel number accounting, control of degausser equipment, and backup capabilities.

FUNCTIONAL LOCATIONS OF VULNERABILITIES

The functional locations of vulnerabilities were analyzed for the 375 cases. Data and report preparation areas and computer operation facilities—the physical locations with the highest concentration of manual functions—were the most vulnerable locations.

Nine primary functional locations of vulnerabilities emerged from the analysis.

(1) Data and Report Preparation Facilities—These were the locations of 120 cases. Areas included key-to-tape/disk/card data conversion, computer job setup, output control and distribution, data collection, and data transportation. Input and output areas associated with on-line, remote terminals are not included here.

(2) Computer Operations—These were the locations of 95 cases. All functional locations concerned with operating computers in the immediate area or rooms housing central computer systems are included in this category. Detached areas containing peripheral equipment cable-connected to computers and computer hardware maintenance areas or offices are also included. On-line remote terminals (connected by telephone circuits to computers) are not included here.

(3) Areas Without EDP Functions—Forty-four cases occurred in non-EDP locations. Many cases involved business decisions in which the primary abusive act occurred in non-EDP areas such as management, marketing, sales, and business offices.

(4) On-Line Terminal Systems—These were the locations of 33 cases. The vulnerable functional areas are within on-line computer software operating systems where acts occur by execution of programmed instructions such as are generated by terminal commands.

(5) Programming Offices—These were the locations of 27 cases. This includes office areas where programmers produce and store program listings and documentation.

(6) Data Preparation and Output Report Handling Areas for On-Line Terminals—Fourteen cases occurred in these locations. This category includes the same functions identified in (1), data preparation but is associated with on-line terminals rather than computers.

(7) Magnetic Tape Storage Facilities—These were the locations of 12 cases. Areas included in the category are tape libraries and any storage place for tapes containing usable data. This does not include temporary or short-term storage of tapes

in tape-drive mounting areas. The latter are included in categories (2), computer operations, and (1), data preparation.

(8) On-line Terminal Operations Areas—These were the locations of ten cases. This category is the equivalent of (2), computer operations, but is in on-line terminal areas.

(9) Central Processors—These were the locations of seven cases. These functional areas are within computer systems where acts occur in the computer software operating system (not induced from terminals).

SAFEGUARDS AGAINST COMPUTER ABUSE

A computer-dependent organization intent on optimizing resource expenditure for computer security can profitably use the results of this study against known and reported types of computer abuse. In general, priorities for safeguards should be established in the following order:

(1) The most important priority, by far, is safeguarding input/output data from disclosure (taking), modification, and denial of use during manual handling in data preparation and distribution.

(2) Secondly, access to sensitive EDP areas should be strictly limited. Particular attention should be given to access by non-EDP employees and non-operational employees (such as programmers) into any operational areas and to the access of operational employees and vendors' employees into operational areas not directly connected with their work. The dangers are primarily vandalism and sabotage.

(3) Computer and on-line terminal operational safeguards are almost as important as access control. The dangers also derive principally from vandalism and sabotage by disgruntled employees in positions of trust.

(4) Business ethics are next in importance to access and operations safeguards. Lack of ethics is a vulnerability found at the highest levels of management and among EDP managers and staff performing or supporting unethical acts for higher management. Accepted ethical standards are needed throughout the computer field.

(5) The next level of concern should be directed at the control of application and operating system programs, including safeguards against unauthorized modification and use, and proprietary aspects.

(6) The specialized and growing problem of preventing access to time-sharing systems by unauthorized users is of next importance.

(7) Finally, the special problem of protecting magnetic tapes in tape storage areas from vandalism and theft must be addressed.

Physical EDP areas of importance for safeguarding appear to be primarily those where manual, operational functions are performed. They are followed by computer systems, then programming offices, and finally magnetic tape libraries. An interesting conjecture is based on the apparent concentration of abuse in the areas of heaviest manual functions. As technical advances eliminate and reduce the size of manual activities, the incidence of abuse could diminish, independent of the degree of security efforts.

REFERENCES

1. Cressy, D., *Other People's Money*, Wadsworth, 1971.
2. Geis, G. (Editor), *White Collar Criminal, The Offender in Business and Professions*, Atherton Press, 1968.
3. Smigel, E. and H. Ross, *Crime Against Bureaucracy*, Van Nostrand Reinhold, 1970.

APPENDIX

RANGE OF ACTS PERPETRATED IN EACH VULNERABILITY: EXAMPLES FROM CASE FILE

Vulnerability/Case Number *

Case Abstract

1. Poor Controls over Manual Handling of Input/Output Data

75327

A keypunch operator in Stockholm, Sweden manipulated payroll data to produce 86 false payroll vouchers payable through the Swedish postal system. She cashed the vouchers at small, remote post offices that had either not received the computer listings of valid vouchers, or did not bother to check the listings before cashing the vouchers. She escaped to South America.

75321

A data control clerk in a bank computer center embezzled $7,200. He diverted and stole checks being processed from a correspondent bank. For each check he then wrote a check for the identical amount on his own checking account and then deposited it in a second checking account, also his own. When his own check turned up for processing he destroyed it and substituted the stolen check. Thus, the check he wrote against his own account was never charged against that account.

* The first two digits of case numbers identify year of occurrence. The third indicates the type of loss (1=vandalism; 2=information or property fraud or theft; 3=financial fraud or theft; 4=unauthorized use or sale of services). The last digit or two digits is a sequence.

7524

Three oil company employees are alleged to have conspired to manipulate computer stored data and oil tank gauges to indicate a full delivery had been made to a refinery from tankers when only part of the load went into the tanks. The rest of the oil is alleged to have been delivered to two smaller companies. The acts were discovered when an inventory check revealed a discrepancy between the amount of oil in the tanks and the amount supposed to be in the tanks according to computer-stored records.

2. Weak or Nonexistent Physical Access Controls

72112

An EDP employee or a vendor's maintenance engineer did $590,000 worth of damage to computer memory stacks by attacking them with a pointed instrument, probably a screwdriver.

7219

An unknown person poured acid over telephone wires where they enter a building containing data processing equipment.

7522

Twelve persons, including a computer manufacturer's employee, thought to be engaged in international espionage, were caught while taking computer components, maintenance manuals, magnetic tapes and circuit diagrams from computing facilities in Frankfurt and Karlsruhe, Germany. Financial losses were estimated to be $110,000.

3. Weaknesses in Computer and Terminal Operations Procedures

74313

In a large bank fraud in Germany, computer operators prevented bookkeeping entries by activating system interrupts from the computer console. Invoices were prepared by the system without recording the transactions.

70410

A programmer analyst used his employer's computer under nonchargeable software development usage in conjunction with his personal outside consulting business.

6711

A disgruntled computer operator dropped pieces of metal into an IBM 2740 terminal causing electrical shorts, fires and considerable downtime. The operator was discharged and successfully prosecuted.

7427

A computer printer operator was paid to make an extra carbon copy of competitive bidding reports for industrial espionage purposes. The operator was discharged.

7438

A computer operator printed copies of unemployment checks and deleted the copying record from the file. An auditor discovered the $10,000 fraud in a computer audit run. The employee was convicted and given a suspended sentence and five years' probation.

7415

A computer operator working alone at night carried a hand gun because the computer center was in a high crime area. One night out of frustration with the computer he shot it with his gun.

4. Weaknesses in Business Ethics

7523

A customer charged a computer equipment vendor with fraudulently representing the capacity and capability of a computer system and charged that the full system was never delivered and did not have adequate software.

7539

One company made a loan to another company based on certain collateral. The pledging of the collateral was hidden by the company receiving the loan so that it could be reused for new loans. Computer listings containing fake data describing the collateral were used.

7446

A computer dating service was sued because referrals for dates were so few and inappropriate. The new owner of the dating bureau said that no computer was used at that time although use of a computer was advertised.

73216

A convicted criminal filed an appeal against the U.S.A. claiming he was denied a fair trial by admission of computer-produced statistical summaries from which inferences of guilt were suggested. The appeal was denied.

5. *Weaknesses in the Control of Computer Programs*

75315

The manager of a bank computer center made unauthorized changes to the demand deposit accounting program to credit service charges to his and a friend's accounts rather than to the proper bank income account.

72219

The president of a French software company posed as a professor of computer science on a tour in the United States where he collected many free programs offered from more than 200 computer centers. He returned to France and sold copies of the programs he had collected. He was caught and fined 5,000 francs for the sale of one program. No action was taken concerning all the other programs he was selling.

6542

A systems programmer replaced a computer operating system supervisory module with a new one that allowed him access to the area of storage used for the resident operating system.

6521

Accounting clerks programmed and used a program called "fudge". It was run at the end of each month to change account balances until all column totals balanced correctly.

7439

A programming manager in a savings and loan association changed updates to the savings program to ignore withdrawals from his account. He was convicted after he was caught by auditors when he made a keypunch error in an account number.

72216

A computer specialist in the government taxation commission in a foreign country sold copies of program logic documentation describing the controls and checking on deduction claims in income tax forms.

73212

A 19-year-old girl convinced her boyfriend to steal copies of computer programs from his employer, a service bureau. She then attempted to sell them to customers of the service bureau. She was convicted and received a one-year suspended sentence.

6. *Weaknesses in Operating System Access and Integrity*

6611

Students caused disruption of a time-sharing service when they tested the operating system and found a failure in the exception handling of filled physical disk storage.

7441

A graduate student compromised a time-sharing system to convert his terminal to the functional equivalent of the computer console. He performed this act to convince management of the computer facility of the vulnerability of the time-sharing system.

7343

A student wrote a program masquerading as an operating system. When a user attempted to log in, the masquerading program obtained his account number and then declared the system unavailable. The student used the account numbers discovered in this fashion for his own purposes to obtain computer time without charge.

7. *Poor Controls over Access Through Impersonation to Time-Sharing Services*

7525

A time-sharing user discovered he was being underbid by small amounts in contract negotiations where the data was stored in a time-sharing service computer. He concluded that a recently terminated employee retained knowledge of his password. He had the password changed, and the problem disappeared.

74316

A man was convicted in France of counterfeiting bank cash dispenser credit cards. He impersonated a bank official to obtain the personal identification numbers associated with the cards by calling the card number holders requesting them to report their numbers so that new numbers could be assigned.

6842

High school students found a time-sharing user's passwords on discarded terminal printouts. The students used the passwords to obtain unauthorized services.

7344

An unauthorized user of a commercial time-sharing service was found to have used eight hours of computer time by continuing to use a password assigned him only for demonstration purposes. The password had not been purged as scheduled.

8. Weaknesses in Magnetic Tape Control

74212

A mailing house employee was caught in attempts to sell magnetic tapes containing mailing lists to a competitor of the mailing house.

7511

An EDP employee sabotaged his company by erasing magnetic tapes with a degausser.

6212

An EDP employee in a bank destroyed all dividend accounts for shareholders of a large company by destroying the magnetic tapes containing the data with a sharp instrument.

7216

A tape librarian in an insurance company was fired but given a 30-day notice. During that time she replaced most of the magnetic tapes in the tape library with scratched tapes.

NOTE: Any opinions, findings, conclusions or recommendations expressed in this paper are those of the author and do not necessarily reflect the views of the National Science Foundation.

Effective safeguards for computer system integrity

by NORMAN R. NIELSEN, BRIAN RUDER and DAVID H. BRANDIN
Stanford Research Institute
Menlo Park, California

ABSTRACT

This paper reports the findings of a project to identify types of computer system integrity safeguards that would have been effective in preventing, detecting, or mitigating the effects of actual reported incidents of computer system integrity violations. More than 350 cases were analyzed and categorized among one of 26 types of violations and among one or more of 34 types of applicable safeguards. Brief definitions are provided for all categories, and distributions of incidents over the various violation categories and over the applicable safeguards are presented.

The analysis revealed that most safeguards have a surprisingly narrow range of applicability, whether measured by number of cases or by number of violation categories affected. However, much broader violation coverage is possible through use of combinations of small numbers of safeguards. Directions for further research are discussed, including the need to develop measures of violation category importance and to include a consideration of safeguard cost, effectiveness, and operability factors.

INTRODUCTION

The history of computing is marked by periods of specialized concerns, such as those for the development of high level programming languages and time-sharing systems. Currently, there is intense interest in computer security and related topics such as privacy and data confidentiality. A number of research projects have been and are being conducted in such areas as physical security equipment, secure operating system kernels, program certification, data encryption, operations procedures, personal identification, and audit practices. While the developments in all these areas are effective against certain types of problems, no general appraisal has been made of their effectiveness against actual computer security problems.

System integrity is used herein to refer to the related and overlapping concerns of:

- Security (protecting system integrity from compromise)
- Audit (verifying the continued existence of system integrity)
- Recovery (restoring system integrity and operability in the event of the loss of such integrity).

Computer system is broadly defined to encompass not only the computer hardware and software but also the computer facility itself. Thus the operations associated with that facility are included, from the initial capture of input data to the final usage of output information by the user. Within the broad perspective of computer system integrity maintenance, the questions of interest concern:

- The safeguards that the user should implement first
- The areas in which development efforts should be focused
- The directions in which future research should be guided.

This paper reports an effort to identify the types of system integrity safeguards that would have been effective in preventing, detecting, or mitigating the effects of actual reported computer system integrity violations. Although the representativeness of the known cases of system violation relative to the total population of actual violations is unknown, this paper does provide some initial information concerning the types of safeguards applicable to the types of violations that are occurring. The scope of potential threats to computer system integrity is vast; hence if a system integrity maintenance budget is to be allocated meaningfully, attention must be focused on the likely threats and the relevant tools for defending against those threats.

The second section describes the integrity violation information used in this research, the third section the categorizations of violations, and the fourth section the safeguards. The fifth section discusses the relationship of the types of violations being experienced to the various types of safeguards. The directions suggested for further research are indicated in the last section.

CASE FILE OF COMPUTER SYSTEM INTEGRITY VIOLATIONS

The computer abuse case file collected by Donn B. Parker was used as the basis for categorizing both violations and safeguards. This file, described by Parker[1] in 1973, now contains information on more than 350 computer incidents. The cases have been collected over a number of years, with information obtained from newspaper articles and other published reports, from personal contacts, and from a number of in-depth interviews with victims and perpetrators. The information available about an incident ranges from a two-inch newspaper clipping to several hundred pages of transcripts and investigatory notes.

It is suspected that, in the history of computing, there have been far more than 350 cases of system integrity violations. However, most of these incidents, for various reasons, have not been reported and in some cases have been actively suppressed. Hence, small as the 350-case sample may be, it represents the most complete coverage available of the known computer system integrity violations.

Each case in the file was studied, and a one-page summary data sheet was filled out. Exhibit I shows the form that was used for this purpose. The collection of summary sheets then became the primary data base on which the remainder of the investigation was based.

EXHIBIT I—Computer Incident Summary Sheet

Questionnaire Date _____

Analysis Complete □

Analysis Incomplete □

1. Case Number __________
   Source _______________
2. Discovery Date ________
   Event Date/Duration ____
3. Accidental ____________
   Intentional ___________
4. Unauthorized use of authorized facility ________
5. Categories: __________
   ____
   ____
   ____
   ____
6. Type of person(s) involved: ______________
   ____
7. Number involved: _______
8. Intent of perpetrator: ______________________
   ____
9. Success of perpetrator: ______________________
10. Brief description: ______________________
    ____
    ____
    ____
    ____
    ____
    ____
    ____
11. Follow-up information: ______________________
    ____
12. Preventive measures: ______________________
    ____
    ____
    ____

INTEGRITY VIOLATION CATEGORIES

In analyzing the types of system integrity violations against which various types of safeguards might be effective, it is helpful to be able to deal with classes or categories of violations rather than with a long list of specific violations. Accordingly, the violation case file was categorized by type of violation.

Each case was studied and given a brief descriptive label. These labels were then collected, refined, and consolidated to form a tentative set of 26 violation categories. Each case was then re-examined by a different analyst and placed into one of the new categories. Differences in placement between the original assignment and the reassignment were resolved, so that there was consensus over the assignment of each case to one and only one of the violation categories.

It should be noted that the choice of 26 categories is not sacred. The categories simply represent a condensation of the various violations for the convenience of this study. It would have been possible to develop only ten categories or as many as 50 categories, if we had found that helpful.

There is as yet no definitive and universally accepted mechanism for determining the precise violation occurring in a given case. Most cases represent a combination of violations. For example, a person enters a restricted area (e.g., a computer room) without authorization, which can be considered one type of violation. Having gained access, the perpetrator then uses the computer in an unauthorized manner or for an unauthorized purpose, which can be considered a different type of violation. The problem is determining which type of violation was the "real" violation.

As a result of such ambiguities, the assignment of cases to the various violation categories is a subjective process. While the placements we have made may not be suitable for all purposes, the subjective nature of our assignment process does not detract from the value of these data to the study. It must be remembered that the categorization of violations serves only to stimulate the isolation or development of safeguard concepts having practical application to the problems actually being experienced.

After all the cases had been reviewed and categorized, 62 were discarded from further consideration. Most of those discarded cases were eliminated because the available information was too sketchy or so vague that it would have been impossible to identify applicable safeguards. A small number of cases were also eliminated because they closely paralleled other cases for which more extensive information was available.

The distribution of the 293 remaining cases across the 26 violation categories is shown in Table I. A brief definition of each violation category is given in Exhibit II.

It is interesting to note that the two violation categories having the largest number of cases (Direct Change of I/O Data, Adding to I/O Data) refer to activities taking place outside the computer itself. However, lest incorrect conclusions be drawn, it is important to note that the distribution figures represent incidents and not a random sample of the full population of incidents. Thus, the concentration of cases noted above may indicate that a preponderance of violations are of this type or that a greater percentage of the violations in these categories are reported or otherwise become known.

TABLE I—Case Distribution Over Violation Categories

	Number of Cases
1. Application Software Manipulation	29
2. System Software Manipulation	18
3. Contract Mistakes	3
4. Improper Use of Personal Identification	2
5. Misuse of System Authorization	4
6. Destruction of Data	7
7. Unauthorized Copying of Data	15
8. Misuse of Passwords	19
9. Direct Change of I/O Data	39
10. Adding to I/O Data	31
11. Personnel Practices	1
12. Unauthorized Building Access	20
13. Violation of Operating Procedures	20
14. Unauthorized Use of Terminal Area	4
15. Misuse of Communications Equipment	4
16. Management Inaction or Misaction	3
17. Unauthorized Use of Services	21
18. Unethical Behavior	7
19. Software Theft	16
20. Improper Training	1
21. Natural Disasters	5
22. Aura of Computer	5
23. Computer Support of Another Crime	2
24. Accident	9
25. Negligence	4
26. Miscellaneous	4
Total	293

EXHIBIT II—System Integrity Violation Categories

1. Application Software Manipulation—Direct manipulation or change of application programs, in the design, implementation, or maintenance stages.
2. System Software Manipulation—Direct change to or nonstandard use of operating system functions or utilities.
3. Contract Mistakes—Poor contract specifications, permitting integrity violations.
4. Improper Use of Personal Identification—Use of personal identification mechanism (e.g., a badge) by an unauthorized person to obtain information or money.
5. Misuse of System Authorization—An otherwise authorized person performing a legitimate task, but one for which he or she is not authorized.
6. Destruction of Data—Physical or logical destruction of data.
7. Unauthorized Copying of Data—Copying files or other data for personal use or resale without authorization.
8. Misuse of Passwords—Unauthorized use of passwords to gain access to computer system.
9. Direct Change of I/O Data—Alteration of computer input data before its entry into the computer system.
10. Adding to I/O Data—Adding data to the computer input stream or to computer outputs after processing.
11. Personnel Practices—Errors or oversights that result in improper privilege level assignments to staff members.
12. Unauthorized Building Access—Unauthorized building access for theft or vandalism.
13. Violation of Operating Procedures—Authorized persons violating computer room procedures or access controls for theft of hardware supplies or for vandalism.
14. Unauthorized Use of Terminal Area—Unauthorized access to the terminal area or use of terminal equipment in unauthorized ways.
15. Misuse of Communications Equipment—Misuse of communications equipment such as lines, multiplexors, and front-ends but excluding terminals.
16. Management Inaction or Misaction—Failure of management to act or improper action because of lack of understanding of computing.
17. Unauthorized Use of Services—Use of computer services in an unauthorized manner (e.g., without payment).
18. Unethical Behavior—Violation of the "reasonable man" ethical standard.
19. Software Theft—Theft of programs or program documentation.
20. Improper Training—Errors made by personnel receiving inadequate or improper training for their assigned duties.
21. Natural Disasters—Damage arising from earthquake, fire, or explosion, flood, etc.
22. Aura of Computer—Computer used as a "know-all" symbol to cheat or mislead people.
23. Computer Support of Another Crime—Computer used in the planning or in the support of another, possibly non-computer related, crime.
24. Accident—Accidental damage to or destruction of data.
25. Negligence—Destruction of data, supplies, or equipment through negligence of personnel.
26. Miscellaneous—violations that do not fit into any of the above categories.

SAFEGUARD CATEGORIES

In analyzing the types of system integrity violations against which various safeguards might be effective, it is helpful to be able to deal with classes or categories of safeguards rather than with a long list of specific safeguards. In the early stages of analysis, it is more helpful to deal, for example, with a category called "personal identification procedures" than it is to deal with a list of specific safeguards such as "badge with photograph, machine readable badge, handprint, fingerprint, signature, and password." Accordingly, the violation case file was categorized by the types of safeguards that would have been effective in preventing, detecting, or mitigating the effects of those violations.

Each case was studied and given a set of brief descriptive labels. Each label described a safeguard that, had it been applied, would have altered the outcome of the incident. Generally two to four safeguards were identified for each case. These labels were then collected, refined, and consolidated, and a tentative set of safeguard categories was formed.

The safeguard categories were themselves analyzed and organized into a set of four generic categories. Each generic category was re-expanded into a set of carefully defined subcategories. Each case was then re-examined and recategorized using the refined categories. This reassignment of cases resulted in one additional round of refinements before the present 34 subcategories were defined and established.

The four generic categories of safeguards are:

- Management safeguards
- Systems safeguards
- Industrial security safeguards
- Legal and educational safeguards

Figure 1 illustrates the hierarchical organization of the safeguard categories, and Exhibit III provides a brief definition of each. Systems safeguards constitute the principal technical defense against computer system integrity violation. Management safeguards are more conventional and less difficult to implement. Industrial security safeguards are the familiar and well understood "physical security" safeguards. The legal and educational safeguards are essentially longer term measures that apply to society as a whole rather than to the environment of a specific organization.

The determination of helpful safeguards for cases faces subjective problems similar to those encountered in assigning cases to violation categories, although the freedom to specify several safeguards eliminates the problem of specifying the safeguard for multiple violation situations (i.e., a situation in which violation A was committed so as to be able to commit violation B). However, the categorization problem for each specific safeguard proposed still remains, analogous to the categorization problem for each violation. Consider the specification of a test procedure to be used as a safeguard in the installation of operating system modifications. Is such a safeguard more appropriately classified as a management procedure, an operations procedure, or a software interface procedure? Thus, a large degree of subjectivity exists in the selection of applicable safeguard categories for cases.

Table II shows distribution of cases across the various types of safeguards that might have been applicable in defending against the violation that occurred. Note that the total number of applicable safeguards is 738, making an average of 2.5 safeguards for each of the 293 cases. It is interesting to note that the two types of safeguards identified as being applicable to the largest number of cases (Audit Procedures, Data

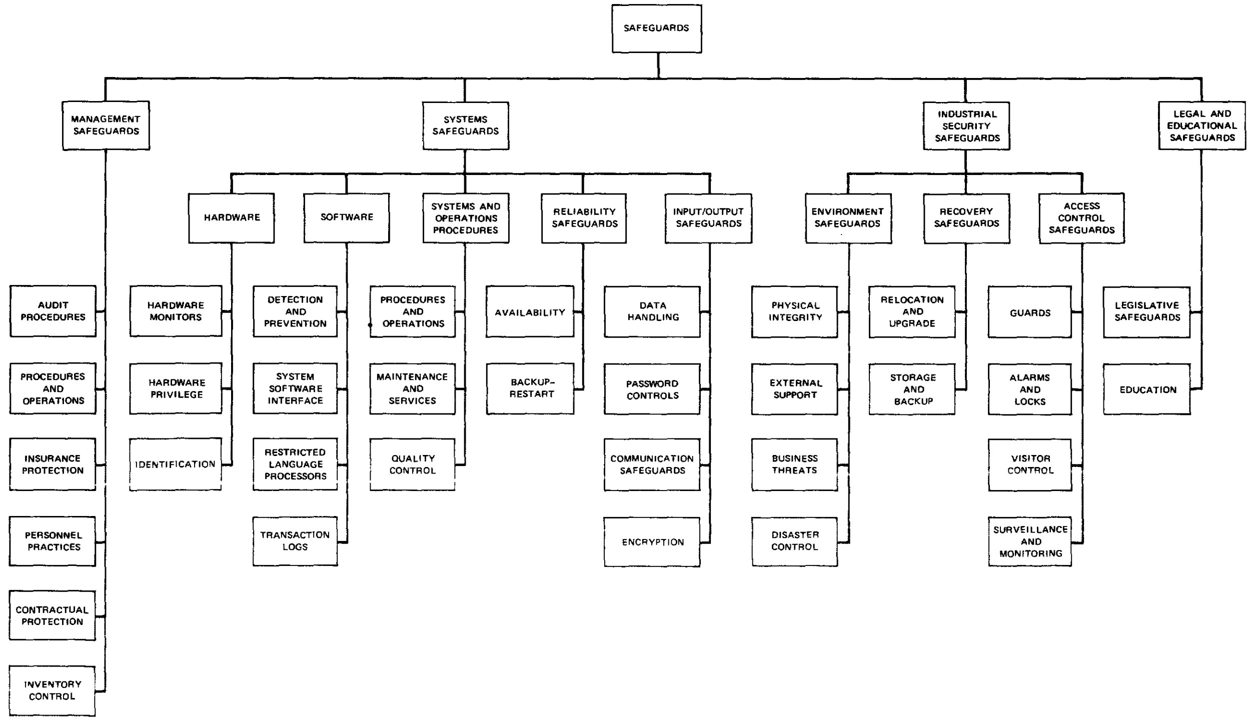


Figure 1—Safeguard schematic

EXHIBIT III—Protective Categories

MANAGEMENT SAFEGUARDS

- Audit—Use of internal and external audits to validate the EDP system.
- Procedures and Operations—Establishment of management procedures that define and enforce operational procedures.
- Insurance Protection—Maintenance of adequate EDP insurance protection.
- Personnel Practices—Investigation of new employees, monitoring of anomalous behavior, and use of effective dismissal techniques.
- Contractual Protection—Use of contracts that address deliverables, specifications, and liabilities.
- Inventory Control—Identification and control of all computational resources and hard copy forms.

SYSTEMS SAFEGUARDS

Hardware

- Hardware Monitors—Use of independent devices for measuring system activity.
- Hardware Privilege—Use of hardware to control access to and use of system resources.
- Identification—Use of hardware devices to identify equipment and people accessing a computer system.

Software

- Detection and Prevention—Use of software to monitor and check program accesses to I/O programs, utilities, and special hardware.
- System Software Interface—Use of software controls to monitor and limit references to operating system components and system utilities.
- Restricted Language Processors—Development and use of families of language processors (and loaders) with increasing levels of privilege.
- Transaction Logs—Use of serialized logs to record transactions, log-ons, I/O, and detected unauthorized accesses.

Systems and Operations Procedures

- Procedures and Operations—Identification of work responsibilities, separation of responsibilities, procedures for handling data, and increased sensitivity to security during abnormal times.
- Maintenance and Services—Use of procedures to ensure timely preventive maintenance and good quality control procedures for software maintenance.
- Quality Control—Use of stringent testing procedures for operating system software and assignment of quality control to separate teams of programmers.

Reliability Safeguards

- Availability—Use of environmental safeguards and architectural configurations that facilitate modular recovery.
- Backup-Restart—Establishment and testing of restart procedures, proper hardware/software backup, and a carefully monitored checkpoint/restart program.

Input/Output Safeguards

- Data Handling—Verification of input data, special handling of extraordinary input, shredding of surplus output, proper storage and backup of data and program files, and limited transmission of output to remote devices.
- Password Controls—Development and enforcement of password procedures, updating of passwords, monitoring of invalid log-ons, and the use of passwords to verify devices and users.
- Communication Safeguards—Establishment of secure communications, hardwired lines, and use of intelligent front-end processors to supplement mainframe coding.
- Encryption—Encryption of sensitive files, including data, password, and accounting files.

INDUSTRIAL SECURITY SAFEGUARDS

Environmental Safeguards

- Physical Integrity—Use of procedures to protect the physical environment of the facility, the use of UPS, and the housing of computing facilities in structurally secure buildings.
- External Support—Establishment of relationships with local police and fire agencies as well as monitoring vendor and other outside personnel in a computing facility.
- Business Threats—Management procedures for anticipating potential threats from competitive and other forces.
- Disaster Control—Establishment of provisions for reacting to natural disasters, e.g., drainage in the event of floods.

Recovery Safeguards

- Relocation and Upgrade—Use of detailed procedures for maintaining system integrity during hardware upgrade or system relocation and comprehensive testing of all system modifications.
- Storage and Backup—Enforcement of operational procedures for storing systems and data backup and documentation in off-site vaults, and the logging of all vault traffic.

Access Control Safeguards

- Guards—Use of building guards to control building access, monitor visitors, and to patrol restricted areas.
- Alarms and Locks—Use of alarms to detect unauthorized entry and locks to limit traffic.
- Visitor Control—Use of proper identification procedures for all persons in the facility, validation of the purpose of all visitors, and investigation of all packages moving in and out of a computing facility.
- Surveillance and Monitoring—Use of surveillance and logging equipment to monitor activities in and around the computing facility.

LEGAL AND EDUCATIONAL SAFEGUARDS

- Legislative Safeguards—Formulation of civil and criminal codes that aid in apprehension and recovery in the event of violations.
- Education—Education of computer practitioners, the public, and law enforcement authorities, curricula developments in protective procedures, and improved professionalism.

Handling) are two areas that are not commonly treated by security research efforts aimed at developing new tools and techniques. It is also interesting that none of the reported violations could have been aided by the application of system availability safeguards. A third observation concerns the large number of cases for which it was judged that some form of procedural development would have been effective, as opposed to some type of hardware or software tool or technique.

Care must be exercised in drawing conclusions from Table II, since the underlying cases do not necessarily represent a random sample of the full population of computer system integrity violations. Thus, the large number of cases for which some type of audit procedure would have been helpful may indicate that a large percentage of the violations actually occurring could have been affected by the application of appropriate audit procedures, or it may only indicate that a large proportion of cases in which audit safeguards were lacking are reported or publicized.

TABLE II—Case Distribution Over Safeguard Categories

Management Safeguards	
Audit Procedures	97
Procedures and Operations	36
Insurance Protection	1
Personnel Practices	11
Contractual Protection	11
Inventory Control	22
Systems Safeguards	
Hardware	
Hardware Monitors	1
Hardware Privilege	9
Identification	17
Software	
Detection and Prevention	63
System Software Interface	14
Restricted Language Processors	11
Transaction Logs	26
Systems and Operations Procedures	
Procedures and Operations	52
Maintenance and Services	2
Quality Control	37
Reliability Safeguards	
Availability	0
Backup-Restart	2
Input/Output Safeguards	
Data Handling	120
Password Controls	38
Communication Safeguards	6
Encryption	13
Legal and Educational Safeguards	
Legislative Safeguards	23
Education	7
Industrial Security Safeguards	
Environment Safeguards	
Physical Integrity	27
External Support	3
Business Threats	6
Disaster Control	22
Recovery Safeguards	
Relocation and Upgrade	2
Storage and Backup	15
Access Control Safeguards	
Guards	18
Alarms and Locks	8
Visitor Control	3
Surveillance and Monitoring	15

Many interrelationships exist between the safeguard categories depicted in Figure 1. However, these relationships are not shown, since their number is so large as to confuse the categorization. Thus, for example, two well-known interrelationships (between Hardware Monitors and Detection Software and between Quality Control and System Software Interface) are not indicated.

It is interesting to note the pervasiveness of procedural aspects throughout the safeguard categories. It was this multitude of interrelationships that presented one of the biggest problems in attempting to isolate and define unique safeguard subcategories. On the other hand, this pervasiveness of procedural safeguards is in itself a significant finding.

SAFEGUARD APPLICABILITY

As discussed earlier, Table II presents the distribution of violation cases over the various types of safeguards that might have been applicable in defending against those violations. However, in view of the unknown representativeness of the violation frequencies of the available cases, it is important to measure safeguard applicability by means other than simply number of relevant cases.

An important alternative measure relates to the range of types of violations against which a safeguard is applicable. Such a measure provides insight into the types of safeguards that potentially have the greatest impact and that should be given prime attention when planning for system integrity maintenance. Accordingly, the safeguards judged applicable to each case were aggregated by violation category. The number of cases in each violation category for each category of applicable safeguard is presented as a matrix in Table III. Several observations should be made relative to these data.

First, there is a tendency for safeguards to cluster, that is, for the safeguards in a set of subcategories to apply to the same types of violation categories. Second, the range of violation types to which a given type of safeguard applies is fairly restricted. Only two violation categories (Data Handling, Detection and Prevention Software) are applicable to at least 50 percent of the violation categories. However, combinations of categories can increase the range of coverage. For example, nearly 70 percent of the violation categories are covered if all the procedural safeguards are considered together. This finding supports the observation discussed earlier about the pervasiveness of procedural safeguards encountered in the formulation and definition of safeguard categories.

As expected, the industrial security safeguards have a very narrow span of applicability. Nominally, Table III shows that some safeguard from this generic category is applicable to cases from 11 violation categories. However, if violation categories having only one safeguard subcategory applicable to but a single case are eliminated from consideration, then the industrial security safeguards apply to only 5 of the 26 violation categories.

It is possible to obtain greater violation category coverage by combining safeguard subcategories. For example, the two management safeguards, Audit Procedures (11 categories) and Procedures and Operations (8 categories), combine to cover 15 categories. The two systems safeguards, Detection and Prevention (13 categories) and Procedures and Operations (10 categories), combine to cover 15 categories. Even Data Handling, with the broadest coverage of any safeguard (17 categories), can be extended to 20 categories when combined with Password Control (10 categories). These types of combinations are very likely to occur in practice because of the natural relationships between safeguards.

Despite the aforementioned overlap in violation category coverage by the various safeguard subcategories, a surprising finding was the lack of broad applicability of the subcategories within a single violation category. For example, in only 5 of the 11 applicable violation categories does Audit Procedures apply to at least 50 percent of the cases in a category. Data Handling applies to a majority of the cases in

TABLE III—Safeguard-Violation Matrix

	Violations								
Safeguards	Application Software	System Software	Contract Mistakes	Personal Identification	System Authorization	Destruction of Data	Copying of Data	Passwords	Direct Change of I/O Data
Number of Cases	29	18	4	2	4	7	15	19	39
Management safeguards									
Audit procedures	20	1				1	2	2	23
Procedures and operations	5								2
Insurance protection									
Personnel practices	1	1				4			
Contractual protection			3						
Inventory control	1	1						1	3
Systems safeguards									
Hardware									
Hardware monitors	1								
Hardware privilege	2	6						1	
Identification	4	3		2	1			1	1
Software									
Detection and prevention	4	6		1	1		6	7	13
System software interface	3	2						5	1
Restricted processors		9				1			
Transaction logs	4	6					6		4
Systems and operations procedures									
Procedures and operations	8	2				1	4	1	11
Maintenance and services									
Quality control	21	6							3
Reliability safeguards									
Availability									
Backup-restart									
Input/output safeguards									
Data handling	4	1		1	1	6	12	4	32
Password controls	4	6		1	1		1	17	
Communication safeguards		3							
Encryption		2					3	7	
Industrial security safeguards									
Environment safeguards									
Physical integrity									
External support									
Business threats									
Disaster control									
Recovery safeguards									
Relocation and upgrade									
Storage and backup									
Access control safeguards									
Guards									
Alarms and locks									
Visitor control	1								
Surveillance and monitoring									
Legal and educational safeguards									
Legislative safeguards		1				2	4		
Education		1							

only 9 of its 17 applicable categories, while Detection and Prevention applies to a majority in only 2 of its 13 categories.

This shallowness of coverage supports the findings from Table II showing only two safeguard categories applicable to as many as 35 percent of the violation incidents (Audit Procedures, Data Handling), only two additional categories applicable to as many as 20 percent of the cases (Detection and Prevention, Procedures and Operations—Systems), and only three additional categories applicable to as many as 10 percent of the cases (Password Control, Quality Control, Procedures and Operations—Management).

The research project to date has utilized applicable violation categories as a measure of breadth of safeguard applicability. While this measure has certain advantages over a measure such as number of applicable cases, it also has certain disadvantages. For example, measurement of range by number of applicable violation categories makes an implicit assumption that all violation categories are of equal importance, an assumption unlikely to hold in practice.

Thus, in drawing conclusions about the "most applicable" types of safeguards, it is important to consider the likelihood of violations occurring in a particular category as well as the likely damage that would result from a violation in that category. A set of weights must be developed for the violation categories, based on both risk of occurrence and risk of loss on occurrence, so as to permit safeguard category coverage to be adjusted by the relative importance of each category. Furthermore, safeguard selection must be based on far more than just potential applicability. Cost, effectiveness, operability, and like factors must be considered as well. Extensions into these areas are part of the planned course of the research project.

TABLE III (Continued)

	Violations								
Safeguards	Adding to I/O Data	Personnel Practices	Building Access	Operating Procedures	Terminal Area	Communications Equipment	Management Action	Use of Services	Unethical Behavior
Number of Cases	31	1	20	20	4	4	3	21	7
Management safeguards									
Audit procedures	26		1				1	18	
Procedures and operations				5	1		3	15	2
Insurance protection									
Personnel practices		1	1	2					
Contractual protection								1	2
Inventory control	9			4					
Systems safeguards									
Hardware									
Hardware monitors									
Hardware privilege									
Identification	1				2	2			
Software									
Detection and prevention	10					2		8	3
System software interface	1	1	1						
Restricted processors						1			
Transaction logs	2					1		1	
Systems and operations procedures									
Procedures and operations	18								
Maintenance and services			1	1					
Quality control	2	1	1					1	1
Reliability safeguards									
Availability									
Backup-restart									
Input/output safeguards									
Data handling	25			4			3	7	3
Password controls			1		3	3		1	
Communication safeguards						3			
Encryption						1			
Industrial security safeguards									
Environment safeguards									
Physical integrity			9	1					
External support			1	1		1			
Business threats			6						
Disaster control			10						
Recovery safeguards									
Relocation and upgrade	2								
Storage and backup			3	3					
Access control safeguards									
Guards			14	3	1				
Alarms and locks			6	1	1				
Visitor control			1						
Surveillance and monitoring			3	10	1			1	
Legal and educational safeguards									
Legislative safeguards								1	3
Education					1				2

Nevertheless, the first-cut, coarse approach used thus far is informative and does provide a meaningful base from which to proceed with further work.

Our analyses and the data presented in Table III support many subjective feelings held about effective types of safeguards. Two conjectures that are supported relate to the potential range of application of particular safeguards. (This support is not affected by the uncertainties that exist concerning the representativeness of the cases used relative to the actual population of computer system integrity violations or by the coarse applicability measure used.) First, the data provide no evidence that there is a type of safeguard that is likely to apply to all types of possible system integrity violations. Second, the data provide evidence that there are likely to be few if any instances where a particular type of safeguard will apply to all cases within a given type of violation.

Consequently, from a user's point of view, effort must be directed toward the development of sets or packages of safeguards rather than the development and refinement of single techniques to address one problem area. The breadth of violations encompassed in a single violation category requires that a similar breadth exist in the safeguards applied.

FURTHER RESEARCH

The research conducted thus far and reported herein has focused on the types of computer system integrity safeguards available and on the range of applicability

TABLE III (Continued)

	Violations									
Safeguards	Software Theft	Improper Training	Natural Disasters	Aura of Computer	Computer Support of Another Crime	Accident	Negligence	Miscellaneous	Number of Covered Categories	Number of Covered Cases
Number of Cases	16	1	5	5	2	9	4	4	26	293
Management safeguards										
Audit procedures					2				11	97
Procedures and operations	3								8	36
Insurance protection		1							1	1
Personnel practices	1								7	11
Contractual protection	3	1						1	6	11
Inventory control	2					1			8	22
Systems safeguards										
Hardware										
Hardware monitors									1	1
Hardware privilege									3	9
Identification									9	17
Software										
Detection and prevention	1			1					13	63
System software interface									7	14
Restricted processors									3	11
Transaction logs				1				1	9	26
Systems and operations procedures										
Procedures and operations	4			1				2	10	52
Maintenance and services									2	2
Quality control				1					9	37
Reliability safeguards										
Availability									0	0
Backup-restart			2						1	2
Input/output safeguards										
Data handling	12			2	1			2	17	120
Password controls									10	38
Communication safeguards									2	6
Encryption									4	13
Industrial security safeguards										
Environment safeguards										
Physical integrity			5			9	3		5	27
External support									3	3
Business threats									1	6
Disaster control			4			5	3		4	22
Recovery safeguards										
Relocation and upgrade									1	2
Storage and backup	1					5	2	1	6	15
Access control safeguards										
Guards									3	18
Alarms and locks									3	8
Visitor control	1								3	3
Surveillance and monitoring									4	15
Legal and educational safeguards										
Legislative safeguards	7			3	1			1	9	23
Education				3					4	7

of these safeguard categories relative to the identified types of computer system integrity violations. It has been an exploratory effort aimed at revealing the types of safeguards that would have been effective in detecting or preventing the actual system integrity violations that have been publicly reported. This work is now being expanded to examine the range of applicability of combinations of safeguards relative to the violations themselves (rather than the categories of violations). The identified safeguards will be categorized according to implementation cost, operational cost, and effectiveness. This should aid in identifying promising research areas and types of tools where further development work would be helpful.

A second phase of the research will be oriented toward the development of an effective collection of detection tools. This set of tools will be designed to address the types of violations not more readily addressed by other categories of safeguards. Emphasis has tended to be placed on the prevention of computer system integrity violations; yet there are many situations where rapid, inexpensive detection of violations (or attempted violations) is more effective from a system standpoint than reliance on stronger but more expensive prevention mechanisms. The research findings with respect to safeguard effectiveness, implementability, operation, and cost will be used as the base for the second phase effort to develop a set of detection tools appropriate for computer system integrity maintenance purposes.

SUMMARY

The coarse analysis of more than 350 incidents of computer system integrity violation revealed a surprisingly narrow range of applicability of particular safeguards over the various types of possible violations. This narrowness was observed both in terms of safeguard applicability to the 26 violation categories and in terms of applicability to the cases within any one category. However, since there are many interrelationships between the 34 safeguard categories, combinations of safeguards are a logical development. The study found that combinations of small numbers of safeguards readily extend the range of violation categories covered.

The study also revealed the desirabiilty of extending the work to consider the relative importance of violation categories, to include consideration of safeguard cost, effectiveness, and operability factors, and to examine combinations of safeguards at the case level rather than at the violation category level.

Interesting sidelights arising from the study were the findings that the two largest violation categories (in terms of number of cases) were ones that apply to activities taking place outside the computer proper and that the two most applicable safeguards (in terms of number of applicable incidents) were ones that are not commonly treated by security research efforts aimed at developing new tools and techniques.

ACKNOWLEDGMENTS

This research was supported in part by the National Science Foundation under Grant DCR74-23774. The viewpoints presented are those of the authors and not necessarily those of the Foundation.

REFERENCE

1. Parker, D. B., S. Nycum, and S. Oüra, *Computer Abuse*, Stanford Research Institute, 1973, (NTIS: PB 231 320/AS).

A centralized approach to computer network security*

by FRANK R. HEINRICH and DAVID J. KAUFMAN
System Development Corporation
Santa Monica, California

ABSTRACT

This paper presents an approach to network security at the system design level. Some basic network concepts and major network security threats are outlined. The design approach is described and a brief security analysis is presented. The proposed network structure incorporates data protection devices called network cryptographic devices and a special-purpose processor, the network security center, to control access in the network.

INTRODUCTION

The ever-increasing utilization of computer systems has heightened demand for broader computer service and data management capability. Computer networks are an attempt to meet this demand by organizing many individual computer systems to act as a single, very large system or supracomputer.

The distribution of data processing functions among a set of distinct systems decentralizes the control of data storage and processing. In addition, information must be transmitted between computers and is therefore subject to exposure. These factors complicate the problem of providing a high degree of security assurance in computer networks. Additionally, current emphasis on privacy considerations underlines the need for network security. Thus security must be a major factor in network design.

This paper presents an approach to network security at the system design level. To provide a basis for discussion of this design, a few basic network concepts are first outlined. Some major network security threats are then presented to provide a context for evaluating the system. Finally, the network structure is described and a brief security analysis presented. The proposed network structure incorporates data protection devices called network cryptographic devices and a special purpose processor, the Network Security Center, to control access in the network.

The design in this paper provides a means for centralizing control in computer networks. When global policies toward network access, data storage and processing can be established, this design is quite appropriate. In some instances, however, it may be difficult to develop such global policies. The management at each network site may decide to maintain greater control over local policy and resist centralization. A second approach to computer network security in which control can be more easily distributed, is presented in a companion paper.[1]

BASIC NETWORK·CONCEPTS

In an intercomputer network, a number of computer systems and terminals are linked. The individual computer systems (hosts) and terminals are called network resources. Interconnection of these resources requires functions performed by both hardware and software, but in this section we consider only the logical arrangement of networking functions rather than associating any particular functions with specific hardware devices.

Network resources must be physically interconnected in some manner. That is, facilities must exist to provide data paths between network resources. These facilities, called the communications subnetwork, may take many forms. The communication subnetwork may consist of telecommunications lines, a message switch, or a packet switched network. Regardless of the configuration, however, we will view communications subnetworks as logically equivalent, supplying a means for data to flow from any network resource to any other network resource.

Figure 1 illustrates three layers, or levels, of network functionality. Layer 1 is network resources; layer 2 is connection-oriented functions; and layer 3 is the communications subnet. Network resources can be thought of as *correspondents,* freely exhanging information (i.e., message text) by way of a *carrier* consisting of the connection-oriented functions and the

* The work reported in this paper was supported by the U.S. Department of Commerce, National Bureau of Standards contract #5-35934.

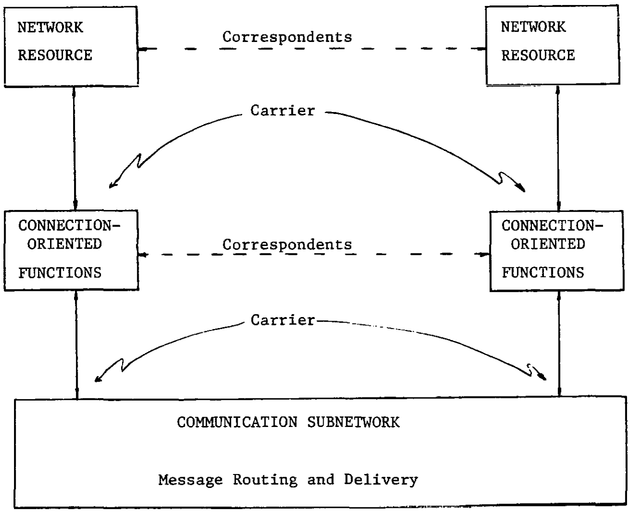


Figure 1—Layers of network functionality

communications subnetwork. The connection-oriented functions at different locations are, in turn, correspondents, exchanging information concerning the state of message pipelines via their carrier, the communications subnetwork. We refer to correspondents as being (logically) above the carriers.

The actual content of the communication between correspondents is not of concern to lower layers (the carrier). Within a carrier, control messages may also be exchanged which are of no concern to higher level correspondents.

Countering the network security threats discussed in the following section will require introduction of additional network functions. These new functions will not alter the logical relationship between the three layers already presented, but will necessitate the addition of a new functional layer.

NETWORK SECURITY THREATS

With privacy statutes being enacted, security vulnerabilities are a serious concern. Yet networks present formidable security problems due to the multi-user, multi-resource, multi-system environment. Physical and procedural controls have proven to be particularly inadequate in such geographically distributed systems. Primary security threats to intercomputer networks are:

1. *Threats to Network Communication*—Network communications are susceptible to several major security threats. Penetrators may tap communication lines or network devices outside of physically secure facilities. Tapping of communications may result in unauthorized exposure of sensitive information or alteration of message text. A penetrator may record legitimate messages and replay them at a later time in order to spoof a network resource. Spoofing could also be accomplished by generation of spurious, but apparently legitimate messages. Misrouting and subsequent misdelivery of messages, either accidentally or maliciously, may result in unauthorized disclosure of sensitive information.

2. *Counterfeit Network Resources*—Network penetrators may be able to utilize counterfeit network resources. A bogus terminal or host computer may be made to appear as a legitimate source or destination of network messages. Without mutual authentication of network resources, uncontrolled use of the network may be obtained by those who would normally not have access to the network.

3. *Forged User Identification*—A penetrator may gain network privileges by forging the identity of a valid user. Of course, this same threat applies to a single computer system. In a network, however, a penetrator may capitalize on a domino effect. A penetrator may use a forged identity to compromise a single host with poor security controls. Other network resources may then be compromised if they, in turn, trust the user's identity as established by the compromised host.

4. *Unauthorized Access by Legitimate Users*—Legitimate network users may gain unauthorized access to host computers, data files, programs, etc. A malicious user may take advantage of unauthorized access to delete or modify data files or programs, or even subvert an entire host computer system. Furthermore, sensitive or private information may be subject to unauthorized browsing.

If each of the host computer systems which make up the intercomputer network were secure when operated separately, the security threats of forged user identification and unauthorized access would be eliminated. Separate network countermeasures for these threats would then be unnecessary. Mechanisms might still be included to relieve each host of the operational burden of implementing identification/authentication mechanisms and to provide a single unified network access protocol increasing user convenience when accessing various network sites. However, no secure general-purpose computer systems exist today. Furthermore, it is doubtful such systems will be widely available for a long time. Thus, network mechanisms must be developed to protect network communications and to avoid increasing compromise threats to hosts because those hosts are linked in a network.

SYSTEM DESCRIPTION

This section presents a system level design of a secure intercomputer network as illustrated in Figure 2. The design incorporates cryptographic devices which

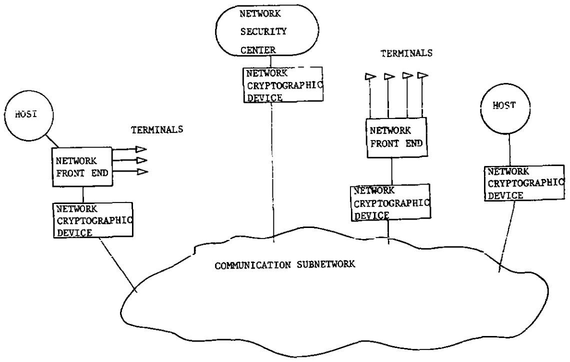

NETWORK SECURITY CENTER
NETWORK CRYPTOGRAPHIC DEVICE
TERMINALS
HOST
TERMINALS
NETWORK FRONT END
NETWORK FRONT END
HOST
NETWORK CRYPTOGRAPHIC DEVICE
NETWORK CRYPTOGRAPHIC DEVICE
NETWORK CRYPTOGRAPHIC DEVICE
COMMUNICATION SUBNETWORK

Figure 2—System level design

encipher data (i.e., transform data in order to conceal its meaning) and decipher data (i.e., reverse the encipher process to render data once again intelligible).[2] This transformation is based on a secret parameter called a Key. The cryptographic devices provide an additional layer in the logical structure of the network.

The design also incorporates a new network resource called a Network Security Center (NSC), which is based on Branstad's concept of a Network Agency.[3] Connections between network resources are permitted only when authorized by the NSC, based on stored access control information. This control is enforced by the network cryptographic devices which will form cryptographic links only when instructed by the NSC.

The network shown in Figure 2 contains Network Front Ends (NFEs). An NFE is a processor which implements connection-oriented functions for a set of terminals and hosts. A network, which adheres to the secure design, can be built without NFEs. NFEs do have operational advantages, however, and are being considered for use in many future networks. Thus, we address their role in network security.

An example may clarify the functioning of the NSC and network cryptographic devices.

A user (U) at a terminal, desires access to a process (P) at a distant host (H). Before being connected with H, the user must carry on a dialogue with the NSC. During this dialogue, U must identify himself and supply additional information, such as a password, to authenticate his identity. U then requests access to host H. The NSC verifies the user's identity. If the user's identity is valid, the access request is checked, otherwise access to H is denied. The NSC uses previously stored access control information to determine if U is permitted access to host H. If the access control information indicates that the access request is legitimate, the NSC will initiate establishment of a logical connection between U and H.

The scenario is similar to that of a user attached directly to a host with an access control mechanism. In that sense the network appears to the user as a single large system.

All messages in the User-NSC dialogue are enciphered and deciphered by cryptographic devices attached to the terminal and to the NSC. Each network cryptographic device has the capability of protecting such dialogues with the NSC. Creating a connection between U and H requires that a new key be established in the cryptographic devices at U's terminal and at H. When the cryptographic devices begin to use the new key they can communicate, forming a cryptographic link between U and H. User U may then initiate formation of a message pipeline to host H via the connection-oriented functions. This connection authorization protocol is similar to that described by Branstad.[2,3]

CRYPTOGRAPHIC DEVICES

There are two main types of cryptographic devices utilized in this design. One is the cryptographic device at the NSC called the master cryptographic device. The other type is attached to each of the other network resources, and is called the slave encryption device.

Slave encryption devices can accept new keys from a remote location. If attached to a single terminal, a slave cryptographic device need maintain only one new key. If attached to a host or NFE, a slave cryptographic device must be able to maintain several new keys in order to support each of the multiple logical connections with a distinct key.

The master cryptographic device must be able to encipher and decipher messages to and from each of the slave cryptographic devices. The master cryptographic device manages establishment of new keys at the slave cryptographic devices.

Both the master and slave cryptographic devices distinguish message headers from message text. Headers must remain in the clear so that the communication subnetwork has sufficient control information to route and deliver messages. Only message text will be enciphered and deciphered.

These devices should make use of the National Bureau of Standards (NBS) Data Encryption Algorithm, which has been proposed as a Federal Information Processing Standard.[4] Several characteristics of this algorithm make it well suited for use in network cryptographic devices:

1. The secrecy of the transformation is dependent only on the secrecy of the key, not on the secrecy of the algorithm.
2. The length of the key is 64 bits, eight of which are reserved for parity. Thus there are 2^{56} potential keys. The key is not so short as to make exhaustive search techniques feasible, yet not so long as to make distribution to a remote device difficult.
3. The algorithm is block-oriented; that is, data

is grouped into blocks of 64 bits which may be enciphered and deciphered independently of any other block. As long as the same key is used, position or time synchronization of encryption with decryption is not required.

Due to routing and transmission differences, message transit time through a network is somewhat variable. Messages may arrive at a destination in a different order than they were sent Using the NBS Algorithm, cryptographic devices can be built which do not require position or time synchronization and are independent of the communication subsystem.

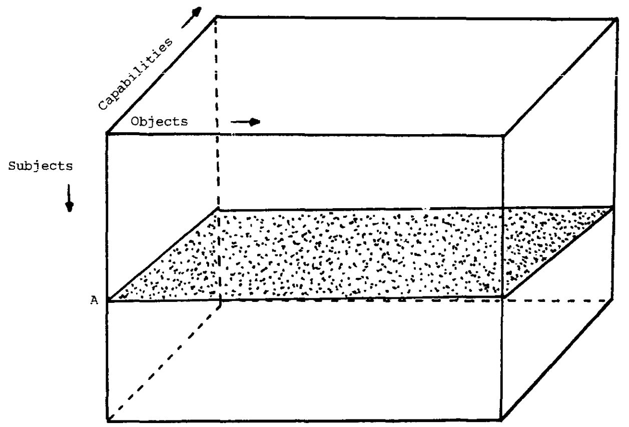
4. When enciphering or deciphering, the change of a single bit in either the key or the input text has an unpredictable effect on the output text. This characteristic has two implications. First, the correct key must be known to make use of (i.e., decipher) enciphered information. Second, alterations to enciphered text cannot produce predictable changes to the corresponding clear text.
5. Analysis of clear/enciphered text pairs does not aid in code-breaking to determine the key used. Penetrators are forced to use impractical exhaustive search techniques for code-breaking.
6. The NBS algorithm is expected to be available as an LSI package. This will provide a low cost, high speed implementation suitable for use in network cryptographic devices.

Network security center

The NSC authenticates the identity of network users and authorizes connections between network resources. When an access request is approved, the NSC must generate a random, distinct encryption key to be distributed to the cryptographic devices at both subject and object. In addition, the NSC will keep audit logs of all access requests, both approved and denied, and will issue appropriate alarms when a suspected penetration attempt is detected.

The NSC must, therefore, maintain a data base which contains sufficient information to verify (authenticate) the identity of users, and sufficient access control information to determine the legitimacy of access requests (access authorization). This data base will not remain static, but will require timely updating. This updating can be accomplished by a security officer at the NSC or by protocols between the NSC and network hosts. Except for authentication of updates, the issues of NSC data base updating are conventional data management system cost and performance tradeoffs and beyond the scope of concern here.

NSC access control information is defined in terms of subjects, objects, and capabilities. A subject is an entity such as a user or a process that can initiate access requests. An object is an entity such as a data file, a process, a host computer system or another network resource that can be the target of access requests. Capabilities are the actions which a subject may perform on an object.



The access control information can be represented by a 3-dimensional space. The shaded plane would contain all information concerning user A.

Figure 3—Access control matrix

A good conceptual model for the access control information is a three-dimensional access matrix[5] as illustrated in Figure 3. On one axis of the matrix are subjects; on another axis are the objects, and on the third axis are the capabilities. Entries in the matrix are boolean values, indicating whether a capability is available to a subject for a given object. This model can accommodate objects to any desired degree of granularity; where granularity refers to the relative size of the subject being controlled. For most systems this matrix is rather sparsely populated, with subjects having access to only a few objects. Thus the actual implementation will use some other more compact and logically equivalent data structure.

Network front ends

A Network Front End (NFE) may interface one or more network resources to the communications subnetwork. The NFE performs the connection-oriented functions on behalf of hosts as well as terminals. The NFE could also provide a user-level command interface for terminals. It is likely that NFEs can reduce the software cost and system overhead normally involved in connecting to networks. A Secure Front End may, in fact, enhance network security, a concept discussed later.

SECURITY ANALYSIS

The system design presented above counters the network security threats. The following discussion analyzes the design approach with respect to the threats presented earlier.

1. *Network Communication Threats*—The characteristics of the NBS data encryption algorithm (and cryptographic devices in general) eliminate many network communication threats. Obviously, line tapping yields encrypted text which cannot be read by a penetrator. Furthermore, alteration of enciphered text can be detected if an error detection field is included in the message. This error check must be enciphered, so that the error check value cannot be predictably altered. Additionally, the check value must be calculated with clear, rather than enciphered, text; otherwise it is possible to alter enciphered text such that the error detection field does not indicate the change. Inclusion of redundancy checks and message sequence numbers within the enciphered portion of the message can prevent undetected message playback or introduction of spurious messages.

The network cryptographic devices used in this design utilize a distinct encryption key for each logical connection between network resources. Therefore, misrouted messages are rendered unintelligible to unauthorized recipients. Currently available "line" cryptographic devices can only be placed on the communication lines, and therefore do not eliminate the threat of misrouting.

Network cryptographic devices with the characteristics required in this design offer greater security assurance than is currently available with existing "line encryption" devices. Although not currently available, network cryptographic devices can be built with current technology.

2. *Counterfeit Network Resources*—The term end-to-end encryption refers to data being enciphered at the source and remaining unintelligible until it is deciphered at its final destination. Network cryptographic devices provide such end-to-end encryption, thereby eliminating the threat of counterfeit network resources. Communication with a bogus network resource is impossible because it would not be attached to a network cryptographic device, or know an appropriate key.

If a network resource, attached to an NFE, is the source or target of network communication, the NFE is responsible for maintaining a proper message pipeline. The NFE must, therefore, guarantee that connections are made with the proper resource. Thus a secure NFE guarantees that the message routing and connection management functions are performed correctly on behalf of attached terminals and hosts.

3. *Forged User Identity*—The NSC requires each user to identify himself and provide information to authenticate that identity. A user's identity is validated before connection to any network resource is permitted. The NSC is a separate tamper-proof mechanism which is not part of a general purpose host computer system. Therefore, the NSC provides a protected environment for the user authentication process, which is less vulnerable than similar mechanisms within a general purpose host.

4. *Unauthorized Access*—The NSC maintains an access control data base that defines all permitted accesses, and must authorize all access requests before a connection between network resource is formed. The NSC is only involved in the initial decision to permit or deny access; an acceptable overhead cost analogous to "opening" a file in most operating systems.

Access requests may specify objects with a varying degree of granularity, but network cryptographic devices can enforce access control only to the granularity of entire network resources. The NSC can, however, pass the results of the access request decision, and any necessary parameters for enforcement, to the host system. The host can then provide the finer granularity of enforcement.

Terminals should not be connected to the network through network hosts. Connection of terminals to the network through general-purpose computer systems needlessly exposes the terminal's communications to security vulnerabilities within the host. Similarly, the hosts are subject to uncontrolled access from the terminals. When terminals are connected directly to the network, on the other hand, all access can be controlled by the NSC. Terminals could therefore either be connected directly to the network with their own cryptographic device (and providing their own connection-oriented functions and message formatting) or be connected to the network through a secure NFE.

SUMMARY AND CONCLUSION

The secure network design outlined here is a centralized management and control philosophy based upon centralized key management. Keys are generated by the NSC and managed by the master cryptographic device. NSC access control decisions are enforced through the use of centralized key management. A companion paper[1] describes an alternative, equally effective approach to network security based upon decentralized key management, and is useful where centralized control is precluded by law, policy, jurisdiction, reliability or practical constraints. In that decentralized approach, all cryptographic devices are identical, but more complex, with each capable of generating keys and relaying keys to other cryptographic devices. The master cryptographic device is eliminated and the NSC is optional.

The network structure described in this paper greatly reduces network security vulnerabilities. The

NSC provides a separate, secure network facility to insure that only legitimate users can access network resources and that only authorized access requests are permitted.

Network cryptographic devices virtually eliminate security threats to network communications and aid in authentication of network resources. Although currently available cryptographic devices do not have the appropriate characteristics, suitable network cryptographic devices can be built with existing technology. Thus, a high degree of cost-effective security assurance can be provided in computer networks with currently available technology.

REFERENCES

1. Kaufman, D. J., *A Distributed Approach to Computer Network Security*, System Development Corporation, SP-3848, May 31, 1976.
2. Branstad, D. K., "Encryption Protection in Computer Data Communications," *Fourth Data Communications Symposium*, Quebec City, Canada, October 1975.
3. Branstad, D. K., "Security Aspects at Computer Networks," *AIAA Computer Network Conference*, Huntsville, Alabama, April 1973.
4. National Bureau of Standards Data Encryption Algorithm, *Federal Register*, March 17, 1975 and August 4, 1975.
5. Lampson, B. W., "Dynamic Protection Structures," *Fall Joint Computer Conference*, 1967.

Computer network cryptography engineering

by HARRISON R. BURRIS
TRW Systems Group, Inc.
Redondo Beach, California

ABSTRACT

A definition of system security and a unified description of encryption methods is presented as background. Alternatives for five major computer network design decisions related to the employment of cryptography with the network are discussed in terms of efficiency (security achieved) and cost.

INTRODUCTION

This paper discusses several design decisions relating to the employment of cryptographic techniques with computer based networks. Cost (hardware purchase prices and effect on performance parameters other than security) and efficiency (security achieved) measures are used for comparisons of alternatives. A definition of security is presented in section two and a method of describing basic types of encryption is presented in section three. The remainder of the paper compares the various network cryptography design alternatives which must be considered in planning efficient secure computer networks.

Given a professionally designed cryptographic algorithm (ignoring considerations of cryptanalytic resistance), the computer network designer should be aware of the impact of decisions concerning the method of employment of the cryptographic techniques upon the overall performance of the system design.

Following the example of Baran,[1] it is assumed that the system attackers are thoroughly familiar with all aspects of the system security design including cryptography and that only the current cryptographic keys are kept secret from the attackers.

There are three possible system security objectives for any computer based system, from single processor to distributed computer network.[2] These are:

Restriction of information to authorized persons.
Protection of system performance (availability and responsiveness).
Restriction of system resources to authorized persons.

One or more of these security objectives may be appropriate to a particular network depending upon the application. Penetration through the network communications system is a high probability threat to all three security objectives and network cryptography is an important system security technique which, depending upon the design decisions made, can either greatly strengthen or weaken the system's threat resistance.

Restriction of information

Restriction of information refers to the objective of preventing unauthorized persons from obtaining the information present in a system. There is nothing as effective as professionally designed cryptography for securing the information content of a communication.

Kahn[4] provides numerous examples of the ease with which "unbreakable" amateur ciphers have been broken by professionals, and restates the maxim that only the skilled cryptanalysts can determine the cryptographic strength of a cipher algorithm. Given professionally designed encryption techniques (hardware or computer algorithms) it is the job of the network engineer to insure that the method of employment does not provide an opportunity to compromise the encryption system.

Protection of system performance

Protection of System Performance refers to all actions taken to prevent system degradation. Degradation of performance is achieved either by causing a system to function with incorrect data so that the outputs are meaningless, or by slowing the system response time until even a correct response is useless. Degradation of performance can be achieved through the communications links by attacking the information being transferred or by monopolizing a sufficient portion of the available communications resources to slow down the system. The manner in which cryptography is employed with a particular network will either reduce the chances of successful attempts at system degradation or may greatly increase the impact of an attack (i.e., it may take much longer to resynchronize an encrypted communications link than one sending clear text).

Degradation of performance is measured as the increase in processing time or computational inaccuracy when the system is under attack over the processing time or computational inaccuracy when the system is operating in a benign environment. This measure should be distinguished from the security cost measured in increased processing time, hardware complexity, or computational inaccuracy of the secure system design over a system performing similar processing but without security capabilities.

An attacker can achieve degradation of performance by attacking the communications network with any combination of four types of attack: (1) Jamming, (2) Playback, (3) Alteration, and (4) Generation. Jamming refers to the introduction of some signal into the communications stream thereby preventing the reception of legitimate transmission. Playback refers to the recording of a legitimate transmission and then reintroduction of the message into the communications stream at some later time. Alteration refers to introducing changes into legitimate transmissions. Generation refers to introducing new messages into the communications stream. Since generation generally implies breaking the cipher before new messages can be produced it will be considered as precluded by the cryptographic algorithms being used in the network.

While cryptography is the principal tool for information protection its contribution to achieving the other security objectives is largely dependent upon the presence of other system security techniques such as Message Sender Identity Verification.[5] Just as a poorly designed encryption algorithm can cause the compromise of information, a poor application of either amateur or professionally designed encryption can greatly increase the impact of an attack aimed at degrading system performance.

Restriction of system resources

Restriction of System Resources refers to the objective of insuring that the system is used only for intended processing, or from another point of view, that each user pays for the system resources used. As for attacks through the communications links (and hence cryptographic design decisions), threats to this security objective are just another form of degradation of performance. From this view, the unauthorized processing load introduced represents the amount of degradation achieved.

ENCRYPTION METHODS

Most digital cryptography has developed as an aspect of digital communications, and this practical rather than theoretical outlook has resulted in digital cryptography being described in terms of a particular logic implementation. The resulting lack of a clear distinction between the cryptographic principles and the logic implementation presents a formidable barrier in assessing the performance (speed and cost) of a particular algorithm compared to other algorithms with similar properties.

The n character message to be encrypted (plaintext) is represented as a character string $P_1,P_2 \ldots, P_n$ and also as a bit string $B_1,B_2 \ldots, B_\alpha$ where $\alpha = n\beta$ and β is the number of bits per character. The n character long encrypted message (cipher text) is represented as a character string $E_1,E_2 \ldots, E_n$ and as a bit string $Y_1,Y_2 \ldots, Y_\alpha$. The different encryption methods (sometimes called privacy transforms[6,7,8,9]) are categorized according to the manner in which plaintext string P is transformed into the encrypted string E.

There are three major categories of enciphering:[4,8] (1) Transposition, (2) Substitution, and (3) Additive Encoding.

(1) *Transposition*—Transposition enciphers a message by reordering the characters of the plaintext. A transposition cipher is decrypted by reordering the encrypted message according to an inverse of the transform used to encrypt the message.

(2) *Substitution*—Substitution enciphers a message by replacing the characters of the plaintext with other characters, perhaps from another alphabet.

(3) *Additive Encoding*—Additive Encoding enciphers a message by combining the bits of the plaintext with the bits of a binary string using the exclusive OR (binary add) function. The encrypted message is decrypted by repeating the exclusive ORing of the encrypted text with the identical binary string.

Transposition

Define a transposition vector, T, of length n such that each value of T controls the transposition of the corresponding character in P into string E according to

$$E_{T_i} \leftarrow P_i \text{ for } 1 \leq i \leq n \text{ and } 1 \leq T_i \leq n \qquad (1)$$

Clearly, $T_i \neq T_j$ is required for $i \neq j$ and $1 \leq i \leq n$ and $1 \leq j \leq n$. The enciphered message is transposed back to the plaintext (decrypted) by the inverse transposition vector, $\overline{T}$, such that

$$P_{\overline{T}_i} \leftarrow E_i \text{ for } 1 \leq i \leq n \qquad (2)$$

where $\overline{T}$ is related to T by $\overline{T}_{T_i} \leftarrow i$ for $i = 1$ to n.

Plaintext messages of length greater than n can be encrypted using a transposition vector T of length n by partitioning P into a series of n long character strings and transposing each separately. The last string of P can be padded to length n with pseudorandom characters without weakening the transposition system.

Substitution

Define the plaintext alphabet, A, as the ordered set of all characters from which characters can be chosen to generate cleartext message strings. Define $\lfloor a \rfloor A$ as the sequence number of character a in the ordered alphabet A.

Define a substitution alphabet, S, such that for every $a \epsilon \{A\}$ there corresponds an $s \epsilon \{S\}$. The correspondence or mapping of A into S is determined by a substitution table F. The substitution table F is defined such that $j = f_i$ indicates a correspondence between A and S such that a_i corresponds to s_j. Table F could in some cases be represented as a function rather than a table which will be done now for brevity. Table I indicates two alphabets A and S whose characters happen to be mutually exclusive.

TABLE I

I	A	S	I	A	S
1	A	S	10	J	2
2	B	T	11	K	3
3	C	U	12	L	4
4	D	V	13	M	5
5	E	W	14	N	6
6	F	X	15	O	7
7	G	Y	16	P	8
8	H	Z	17	Q	9
9	I	1	18	R	0

For an F table defined according to the function $f_i = i$, the correspondence between the alphabets of Table I would be $Y \epsilon S$ corresponds to $G \epsilon A$. For the function $f_i = n+1-i$ with $n = 18$, Table I indicates the correspondence of character $4 \epsilon S$ to $G \epsilon A$.

A plaintext message P is encrypted by substitution on a character by character basis where P_i is encrypted according to

$$E_i \leftarrow S_{f_j} \text{ where } j = \lfloor p_i \rfloor A \qquad (3)$$

The encrypted message string is decrypted using the inverse transform vector F defined so that

$$P_i \leftarrow A_{f_j} \text{ where } j = \lfloor s_i \rfloor S \qquad (4)$$

Substitution ciphers are not explicitly influenced by the length of the plaintext strings.

Substitution ciphers where only one substitution alphabet and one substitution function are defined are called monalphabetic (including $S \equiv A$ except that for this case $f_i \neq i$ is required to avoid an identity transform). Polyalphabetic substitution ciphers[4,10] can be represented as multiple mappings (multiple F's) into a single S, as multiple alphabets controlled by a single F, or as a combination of multiple S's and F's. In order to complicate the cryptanalysis of substitution ciphers, several characters in S are often defined as being equivalent to one character in the plaintext alphabet. These sets of equivalent substitution characters are called homophones.[4]

Additive encoding

An additive encryption system transforms the plaintext bit string B into the enciphered string Y by applying the exclusive OR operation to string B and ciphering string X on a bit by bit basis where

$$Y_i \leftarrow B_i \oplus X_i \text{ for } i = 1 \text{ to } \alpha. \qquad (5)$$

It is the property of the exclusive OR operation that string Y can be decrypted by a repeated application of the transform

$$B_i \leftarrow Y_i \oplus X_i \text{ for } i = 1 \text{ to } \alpha. \qquad (6)$$

Partitioning the encrypted bit string into characters shows a key additive encryption to be equivalent to a polyalphabet substitution.

Plaintext strings longer than the coding string can be encrypted by partitioning the plaintext string into a series of bit strings of length α and encrypting each separately. The last string can be padded (random characters should be used for padding to strengthen the crypto system).

Encryption primitive security considerations

Jamming and Playback are not directed against the cryptographic primitives. For this presentation, generation has been eliminated as a threat (professional cryptographic algorithms), so only alteration remains as an attack directly influenced by the cryptographic primitives embodied in an implementation.

With a transposition cipher the plaintext characters are replicated in the ciphertext string so the attacker can precisely determine the plaintext character that result if a change to a ciphertext character is made. However, since the T vector is unknown, the correct position occupied by each character in the plaintext string cannot be determined. With both substitution and additive ciphers, the exact position of an altered character is known to the attacker but the exact plaintext character represented by each ciphertext character cannot be determined. Numerous highly effective probabilistic attacks can be made against substitution and additive cipher systems.[5] Cryptographic implementations employing multiple transposition and substitution or additive primitives can counter all but brute force alteration attacks since both position and resulting plaintext are unknown.

SYNCHRONIZATION

Synchronization of cryptographic devices is the process by which the encrypter at the sending end and decrypter at the receiving end are kept in step with each other. Three synchronization alternatives exist. The first two are in general use, the third is considered a practical proposal.

(1) *Link Synchronous Encryption*—The term link

synchronous is applied to an employment of cryptographic devices in which a one directional (simplex) point-to-point communications channel is enciphered such that a continuous stream of encrypted characters appear on the communications link and the receiving crypto device (the decrypter) is kept in step with the key of the sending encryption device by counting characters in the received data stream. In order to maintain the continuous character stream when no data is available to be sent, the encrypting end of the link generates a string of pseudorandom padding characters which are switched in and out of the transmission stream as required.

Link synchronous encryption is highly susceptible to degradation of performance attacks, since once the encryption devices are forced out of synchronization (i.e., by jamming) it requires a relatively long period to reestablish the network. Thus, a short duration jamming attack on the part of the attacker can deny use of the communication link for a period much longer than the period of jamming.

Because the communications link is continuously in operation, the network attacker is denied information about the volume and frequency of message traffic. This is called transmission security (TRANSEC).

The costs of this synchronization method are extremely high in terms of encryption equipment and communications resources (radio frequency spectrum or wire lines) required. The upper bound for a fully interconnected n node network is n^2-n links (dedicated frequencies or wire lines) and 2(n^2-n) encryption devices.

(2) *Packet Synchronous*—A packet is a block of characters, which may be either a segment of a message or an entire message, and may or may not be of fixed length. The term packet synchronous will be used to describe methods of synchronization which rely upon the appending of crypto synchronization information to the header of the packet in order to set the decryption device to the appropriate key.

In this mode, packets may be deleted without detection, and playback is possible. These attacks are facilitated by packet synchronization because as long as the synchronization and message text are associated together, both can be sent to a receiver at a later time and still be decrypted correctly. This method does not require the time consuming resynchronization processes of link synchronous systems since each packet carries its own synchronizing information.

Since a network can be established in which each node (and encryption device) recognizes its own address in the packet header, the costs of this method are considerably less than those for link synchronization since dedicated links and encryption devices are no longer required. For an n node network offering fully interconnected routing only n links and n devices are required.

(3) *Clock Synchronous*—Clock synchronous is a term proposed for the following method of encryption synchronization. The use of extremely accurate atomic (Rubidium or Cesium Beam) clocks for achieving synchronization of communications devices has been proven and portable clocks are available.[11] It should be possible to use the same methods to synchronize encryption.

In this mode a clock at each node is used to control the advance of the key. The clock time at which encryption was begun is appended to the message packet and serves as the synchronization information to set the key at the receiving node. If the message is not received within a set time period after encryption, it is rejected. Aside from the reduction of the synchronization bit string (often longer than the data in the packet) to a string just long enough to contain the start time to the required accuracy, the clock synchronous method offers no additional benefits over the packet synchronous mode. An atomic clock is required at each node for the clock synchronous method.

IMPLEMENTATION OF ENCRYPTION DEVICES

Presupposing the cryptographic algorithms are highly resistant to cryptanalysis it is extremely important to insure that the information to be protected is not compromised in some other fashion. It is possible that a circuit failure could result in plaintext being passed through a failed encryption device without detection. It is possible the electromagnetic radiation caused by the encryption device could radiate the plaintext data.[9,12-Ch. 29]

While status indicators or software checks are available to detect the failure of an applique or main processor cryptographic process, these are not nearly as reliable for preventing accidental information release as using an LSI cryptographic device tied to some other critical circuit such as the main processor instruction sequence controller (CPU master clock) so that if the encryption chip failed, the main processor would stop within one instruction!

There are some applications where it is critical that the processing functions be performed even when the security system has failed. In these circumstances, the strong argument for LSI becomes a liability. Bypassing a failed applique is usually accomplished by a switch action or at most replugging a patchboard, bypassing circuits at the LSI or even card level can be a more difficult problem. However, as LSI availability increases, multiple encryption chips could be used with "hot spares" switched in after a failure.

KEYING METHODS

The autokey (or ciphertext autokey) method was developed for use with polyalphabetic substitution ciphers.[4] For a plaintext alphabet of n characters define n substitution functions (and inverse functions)

$F^1,F^2,\ldots F^n$ and $\bar{F}^1,\bar{F}^2,\ldots\bar{F}^n$ each of which specifies a transform of every character in A into S and vice versa. The notation F_j^i is defined for multiple functions where i specifies which substitution function is to be used and j is the input (sequence number) to the function.

Encryption begins by placing an extra plaintext character which indicates the start of the key at the beginning of the plaintext and enciphered strings ($P_0 \equiv E_0$). The plaintext is then enciphered according to

$$E_i \leftarrow S_j \text{ for } j = F^{P_{i-1}}_{\lfloor P_i \rfloor A} \text{ and } i=1 \text{ to } n. \qquad (7)$$

where P_{i-1} determines which substitution function F will be used to transform P_i into E_i. The decryption process is

$$P_i \leftarrow A_j \text{ for } j = \bar{F}^{P_{i-1}}_{\lfloor E_i \rfloor S} \text{ and } i=1 \text{ to } n. \qquad (8)$$

To begin the decryption, recall that $P_0 \equiv E_0$.

It appears practical to extend the autokey strategy to transposition and additive encryption methods. Both of these methods may employ a fixed length transform of n characters at a time. Define a key as a set of transforms for either method with one transform for every possible character in the plaintext alphabet A. Then designate one character position in the n character plaintext string as the position controlling the selection of the transform to be used for encrypting the next n character string. This completely defines an autokeying encryption process. Similar to substitution, decryption is controlled by the key selection character position of the most recently decrypted plaintext string.

Some autokey systems have not been popular for computer network applications because of their tendency to propagate communications errors (or attacker induced changes). A transmission error in one position causes the wrong inverse transform to be selected for decrypting the next character. However, good autokey systems can be devised to be self-synchronizing after an error. Error propagation increases the attacker's leverage for denying the use of the communications resources. However, it makes attacks relying upon the acceptance of an altered message almost impossible. The autokey methods proposed above for transposition and additive systems do not perform as well for detecting alteration, as in many applications it would be unacceptable to wait until receipt of the next message before determining that the system has been penetrated. Even this after-the-fact indication would not be present if the attacker were careful to avoid altering the character position controlling selection of the next key. Error propagation would occur for these transforms only for the case where the key selecting character was in error.

DISTRIBUTION OF KEY MATERIAL

The particular key used with a set of crypto-algorithms is usually changed after a stated period of time (the crypto period). This change makes the playback of previous messages difficult and increases the work of the cryptanalysts since the statistics collected on the previous key must now be repeated. Key material is also changed or updated whenever it is suspected that the previous key has been compromised.

There are two principal alternative methods which can be used to distribute the new keying material: (1) the keys can be transmitted over the network, or (2) the keys can be distributed by an independent distribution system.

Manual key distribution systems require extra storage capability for keys, special handling, such as couriers, and require a relatively long time (manual action) to insert the new key. The strength of this method, which is the standard practice today, is that it requires a physical compromise of the key material in order for the encryption system to be broken during key distribution. Distribution of new key material over the network would have the advantage of changing crypto periods at machine (computer and communications) speeds. However, if the network is used to distribute new keys using an old key, compromise of this key potentially compromises all subsequent keys.

EXTENT OF ENCRYPTION

The network design decision to encrypt all messages (encrypt-all) or to encrypt only messages requiring protection (encrypt-select) must be considered by the network architect. Enciphering equipment is expensive, especially for remote terminals that do not handle information that must be protected. This observation has led to the development of several networks where some nodes are interconnected by encrypted communications and other nodes of the network are interconnected by unencrypted communications.[1,12] Another implementation with similar properties is a network where information requiring protection is sent enciphered (ciphertext) and information not requiring protection is sent unencrypted (cleartext). The advantages of these cleartext/ciphertext systems are that (1) low cost, portable nodes can be used where encryption is not necessary and (2) the network can be easily reconfigured for emergency transmission of priority information even if some encryption devices have failed.

The advantages of encrypt-select designs are transitory at best, since LSI encryption devices should soon eliminate the cost/size/power objections to encryption of even the lowest priority remote terminals. More importantly, the increased complexity in message switching software and encryption status checking hardware in the processors far outweighs the ease of reconfiguration achieved. Once total encryption becomes feasible these bimodal networks will probably cease to exist.

SUMMARY

Several network design considerations involving cryptography were discussed in terms of security and cost. In nearly every case each design alternative traded increased security from one form of attack at the expense of increased susceptibility to another attack. The particular choice of a method therefore is left to the designer depending upon the intended application of the network.

REFERENCES

1. Baran, P., *On Distributed Communications: IX. Security, Secrecy, and Tamper-free Considerations*, Doc. RM-3765-RP, Rand Corp., Santa Monica, Calif., August 1964.
2. Burris, H. R., "System Security Planning Guide: Concepts and Projects," *System Security Series No. 1*, Office of the Project Manager Army Tactical Data Systems, Ft. Monmouth, N. J., October 1972.
3. Girsdansky, M. B., "Cryptology, the Computer and Data Privacy," *Computers and Automation*, Vol. 21, No. 4, April 1972, pp. 12-19.
4. Kahn, D., *The Codebreakers*, The Macmillan Company, New York, 1967.
5. Burris, H. R., "Performance Comparison of Authentication and Serialization Message Sender Identity Verification Techniques," *System Security Series No. 4*, Office of the Project Manager Army Tactical Data Systems, Ft. Monmouth, N. J., February, 1973.
6. Carroll, J. M. and P. M. McLelland, "Fast Infinite Key Privacy Transformation for Resource Sharing Systems," *Proc. AFIPS 1970 FJCC*, Vol. 26, AFIPS Press, Montvale, N. J., pp. 223-230.
7. Hoffman, L. J., *Security and Privacy in Computer Systems*, Melville Publishing Company, Los Angeles, Calif., 1973.
8. Petersen, H. E. and R. Turn, "System Implications of Information Privacy," *Proc. AFIPS 1967 SJCC*, Vol. 30, AFIPS Press, Montvale, N. J., pp. 291-300.
9. Turn, R. and N. Z. Shapiro, *Privacy and Security in Databank Systems: Measures of Effectiveness, Costs, and Protector-Intruder Interactions*, Report P-4871, The Rand Corporation, 1972.
10. Skatrud, R. O., "The Applications of Cryptographic Techniques to Data Processing," *Proc. AFIPS 1969 FJCC*, Vol. 34, AFIPS Press, Montvale, N. J., pp. 111-117.
11. *Electronic Instruments and Systems*, Hewlett Packard, 1975.
12. Martin, J., *Security, Accuracy, and Privacy in Computer Systems*, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1973.

The application of cryptography for data base security

by EHUD GUDES and HARVEY S. KOCH
The Ohio State University
Columbus, Ohio

and

FRED A. STAHL
Columbia University
New York, New York

ABSTRACT

The application of cryptographic transformations for the purpose of enhancing the security in data base systems is discussed. These transformations have been recognized in the past as a valuable protection mechanism but their relation to data base security has not been identified. The major reason is the lack of a suitable data base model for investigating the questions of security and cryptography. A multi-level model of a data base is presented in this paper. This model helps to understand the connection between the data base structure and the cryptographic transformations applied to the data base. It is shown that cryptographic transformations can be applied between the different levels of the data base. Several types of these transformations are identified and the possible ways of using and controlling them are also discussed. The multi-level model can provide a useful framework for further research in the area of cryptography and data base security.

INTRODUCTION

The technical problems associated with providing protection for information in a shared computer environment have received considerable attention in recent years. Petersen and Turn discussed a broad spectrum of these problems in their article on "System Implications of Information Privacy."[14] Subsequently, there have been numerous endeavors attempting to amplify and find reasonable solutions to these problems.[6,7,9,11,12]

Concurrent with the developments in the area of protection has been the very important research in data base systems. This is due principally to the growing size and complexity of existing data bases. Research in data security and research in data base systems have only been combined recently. Most data security models,[7,11] use a unified approach in which all components of the computer system (i.e., objects) are viewed as being on the same level. A data base system is viewed as much more complex than the traditional file system and therefore its security problems are more complicated. In a data base, protection may be required from the file and record level down to the field level. User protection requirements of a data base are more complex than protection of a full file (or segment) and complex protection specifications based on boolean expressions may be needed. A unified approach is, therefore, not suitable for dealing with the security problems in data base systems. Our view is that the security problems of data base systems have to be dealt with separately from the security problems of operating systems. (A similar view is held by Minsky.[13]) However, since a data base system uses the operating system services, the two problems cannot be completely disconnected.

Since we are interested in data base systems, our assumption is that the hardware and the operating system are correct and secure. This is not as strong an assumption as it may seem at first glance. Since we have removed the problems of files and data bases from the operating system domain, the operating system becomes smaller and easier to verify. Clearly, the complexity of the protection problem in the data base system increases.

In this paper we concentrate on the security problems of data base systems. We develop a model of a data base which allows the incorporation of several protection mechanisms. The goal is to define a structural model of a data base in which known protection mechanisms can be applied, and their dependency on the structure of the data base can be understood. In this paper we are interested in one protection mechanism called *cryptographic transformations*. The value of these transformations as a means of protection has been well established[14] and some research was done on their application in file systems.[19] This model will help us to understand where and how to apply these transformations, and how these transformations can be used in conjunction with other protection mechanisms, in a data base system.

CRYPTOGRAPHY AND DATA BASES

Cryptographic transformations have been recognized long ago to be an effective protection mechanism in communication systems. In the past they have been used mainly to protect information which is transferred through communication lines. However, as Peterson and Turn[14] pointed out, they can be used as an effective counter-measure against some of the threats to security which exist in computer systems. Among them are: wiretapping, between lines entry, browsing files, physical acquisition of removable files. Although several other articles mention their existence and suggest their application in computer systems (Skatrud,[18] Van Tassel[20]), not much (public) research has been done in this area. In particular, there is no research on the problems of how to apply cryptographic transformations to permanent sharable data bases as opposed to their application to information transferred through communication lines. We call the first form of cryptography *data base cryptography* as opposed to the more popular form of *communication cryptography.*

There are large differences between the constraints put on communication cryptography (see Shannon[15]) and between those put on data base cryptography. Turn[19] enumerated some of these differences. The major ones are: (a) *The problem of selective retrieval*—because files are usually organized so that selective retrieval of records can be achieved, it is very desirable that enciphering (deciphering) of record *i* will not depend on another record *j*. This constraint prevents the use of the popular Vernam Cipher using a pseudorandom number generator of a very large period. Such a generator would be usually used for enciphering large quantities of data (e.g., the whole file) and would have to include more than one record in the enciphering process.* (b) *The long "life" of the data*—data in data bases usually resides there for relatively long periods. Therefore, the very popular method of changing the cryptographic keys very often cannot be applied, since it will require a complete reprocessing of the data base or a large part of it. (c) *The processing problem*—data in files and data bases is stored for processing purposes. It would be very desirable if we could process the ciphered data in the same way we process the "clear" data. The reasons for this are that the system is more secure if only ciphered data is processed and the overhead of enciphering/deciphering every time we access the data is saved. We would like therefore to design "processable" ciphers. Examples of such ciphers will be given later. It can also be shown that in the case of a "processable" cipher, applying the cryptographic transformation on the data item level only, is not secure enough. Given the constraints above, it is clear that the subject of data base cryptography is strongly connected to the subject of data base organization, representation and accessing. None of the current models of data base systems addresses this problem directly. Furthermore, current data base models are not suitable for answering questions related to data base cryptography.

Very few models of security in data bases mention the use of cryptographic transformations. The CODASYL[3] model provides the ENCODING/DECODING clause on the data item level. However, application on the data item level *only* may not be secure enough. The connection to other protection mechanisms is not clear. Hoffman[9] mentions the SCRAMBLE/UNSCRAMBLE procedures, but does not give any details of their use.

We are then faced with the following questions: (a) To which level should the cryptographic transformations belong? To the physical structure, to the logical structure, or to the mapping between them? (b) Should the Data Base Administrator (DBA) have complete control on the cryptographic transformations? Similarly, should the keys for these transformations be part of the system, e.g., in its Data Definition Language (DDL), or should only the appropriate user have some of the cryptographic keys? (c) Should cryptographic transformations preserve or destroy the structure of the data base and what are the advantages and disadvantages of each case? (d) What is the relation with other protection mechanisms? Should they complement each other and how should it be done? The main goal of this paper is to answer the questions above. However, in order to answer them, we need a *framework—a data base model* in which the security problems and their relation to the data base structure are clearly identified. Such a model will be developed. Before we develop this model, we want to review some of the basic concepts in data security.

BASIC CONCEPTS

Looking at the literature on data security, we find some degree of confusion about the basic concepts. We will use the terms *security, protection* and *access control* interchangeably and assign them the following meaning by McCauley,[12] "The process of determining the authorized users of the data base and of determining which access may be permitted and which would be denied." Graham and Denning[7] made a very important distinction between the *protection specification* and the *protection mechanism* in computer systems. The protection specifications are the translation of management privacy views into exact specifications. The

* Of course, VERNAM cipher can be applied to each record separately, probably using different "seeds" for the pseudorandom number generator, but then its security is far from a "one time" cipher.

protection mechanism is the mechanism to execute correctly the protection specifications and to assure that any *protection violation* of these specifications will be detected.

In different systems there exist different protection mechanisms. Most of these mechanisms are composed of two parts: the *protection procedure* and the *protection data*. The protection data is not to be confused with the protection specification. The protection data is data which is internal to the protection mechanism, for example—passwords in the case of the password protection mechanism or tags in Friedman's model.[5] The protection procedure is analogous to the program or procedure in programming systems and it is the coded form of the protection mechanism algorithm. An important example of such a two part protection mechanism are *cryptographic transformations*. In this case, the transformation algorithm is the protection procedure while the cryptographic keys are the protection data. The analogy to programming systems can be carried further as follows:

Programming Systems	*Data Security*
Procedure	Protection Procedure
Temporary Variables	Protection Data
Input Data	Protection Specifications
Output Data	Protection Decision (i.e., Grant or Deny)

As an example for the use of the concepts above, let us look at the protection mechanism in OS/360 file system.[10] The protection mechanism is a password protection mechanism where a file can be accessed if and only if the right password is given by the user during OPEN. The protection procedure then is part of OPEN. The protection data is the list of passwords. The protection specification is the distribution of passwords between users according to the privacy decisions: which user has access to which file. So passwords here has a double role: as the protection data and as the way to express the protection specifications.

In the next section, we will use the concepts defined here in the discussion of the data base model and its relation to security.

A MULTI-LEVEL STRUCTURED MODEL FOR A DATA BASE

The existence of several data base levels is well recognized in the data base "community." Usually a distinction is made between the logical level (structure) and the physical level (structure). A good discussion of these levels and the mapping between them is given in Sibley & Taylor.[17] In the CODASYL model we can distinguish three levels: the Sub-Schema and Schema levels which define the logical structure and the storage level which defines the physical structure. A four level model known as the entity-set model was suggested by Senko, et al.[1] Another four level model, similar in concept to the one we suggest here, was suggested by Sibley.[16] We could have developed most of our notions in the framework of one of these models, however we preferred to develop our own terminology and structure for two major reasons: First it allows us to stress the security point of view which we are interested in. Secondly, our model differs from other models by its recognition of the existence of *more than one physical level in a data base.* In most data base models only one physical level is recognized and this is usually the secondary storage structure. However in most conventional systems (even with virtual memory), data exists physically in more than one medium and this fact is very important from the security and cryptographic points of view. The main idea in this model is that a *data base is composed of several logical or abstract levels which describe data which physically resides in one or more physical media and therefore have one or more physical structures.*

The existence of these physical media is recognized even in the conceptual model of CODASYL.[3, p. 15] In CODASYL three media are identified:

1. The user-working area
2. The system buffers
3. The secondary storage

Therefore we should have logical schema which describe the data structure in each of these levels. Actually in the CODASYL model the sub-schema describes the data in Physical Level 1 while the schema describes the data in Physical Levels 2 and 3.

A FOUR LEVEL MODEL

In the model that we have developed there are *four logical-abstract levels:*

1. User-logical level
2. System-logical level
3. Access level
4. Storage level (also called structured storage level)

Each of these levels can be composed of several sublevels. Corresponding to each logical level there is a physical level which is connected to a physical medium. We will now concentrate on the description of the logical levels and their relation to security.

The *user-logical* level corresponds to the way a user or a user group sees the data base. It is very similar to CODASYL's Sub-Schema with the exception that we do not have the constraint that the user-logical level must be a subset of the system logical level. On the contrary, it might be useful to have complex transformations between the two levels. Usually there are several user-logical level structures in a data base. The *system-logical* level describes the whole logical structure of the data base. It may correspond to CODASYL's Schema with the difference that indexes, directories, and access paths are not part of the system-logical level (they are part of the Schema in CODASYL.) The *access* level describes the directories, indexes and all possible access paths in the data base. The storage level is the result of applying the access level to a particular physical secondary storage device(s) and describes characteristics which are special for these devices. To each logical level corresponds one or more physical levels according to the number of physical media. An example is shown in Figure 1.

In this example the user logical level describes the data as it appears on the user site. The system logical level gives the interpretation to data which appears in memory. And the access and storage levels give the right interpretations to the data which reside on the secondary storage devices. It is conceivable that in the future the secondary storage hardware will do all the access calculations and therefore no access information has to be in memory or described in the system logical level. The main idea is the existence of more than one physical level corresponding to more than one physical media.

The relation of this model to security is discussed in Gudes.[8] The model is shown to be general and to include other known data base security models[5,12] as special cases. The main idea is the *decentralization* of protection mechanisms by the "spreading" of protection specifications and mechanisms through the different levels of the data base. In this paper we are interested with only one protection mechanism—cryptographic transformations. The model will be used as a framework for the application of cryptographic transformations in a data base. As is shown in the following sections, cryptographic transformations are a subset of the transformations between the physical levels of a data base, which is used for protection.

FORMALISM

In order to understand the relationship between the logical levels of a data base and their corresponding physical levels we need some notation. When one looks on a data base as it is represented on secondary storage one sees a sequence of 0's and 1's. These binary digits make sense only when one knows the right *structure, coding* and *interpretation* of the data. One starts with the simple division to data items and then starts to build the more complex blocks of the structure (repeating groups, records, files). The basic concepts, then, for describing a data base is the concept of the *data item*. A *data item* has the following properties:

interpretation (attribute)—what the data item is
length and what it is used
value for.
representation (coding)
address

We denote data items as: d_i

A common case in data bases is that one data item contains one or more properties of other data items. We give two examples:

Example 1	d_1	d_2
interpretation:	Length of d_2	name
value:	5	SMITH

In this case d_1 contains the property: length of d_2.

Example 2	d_1	d_2
interpretation:	units	distance
value:	0 or 1	100

If $d_1 = 0$ then d_2 is in miles
If $d_1 = 1$ then d_2 is in kilometers

Hence d_1 contains the interpretation of d_2. However we need also the interpretation of d_1 in order to know what d_2 is. The interpretation of d_1 is probably documented in some manual which describes the system.

We see then that a set of data items may have *several levels of interpretation.* Some of them are in the data base itself and some of them are only implicit. Each physical level of the data is just a set of data items, where their interpretation is either in some of the data items themselves, or in the logical description of this level, or documented in some manual. More formally, a physical record j on level i is denoted as PR_j^i where this physical record has the address A_j. This physical record is an ordered tuple of data items

$$PR_j^0 = (d_1{}^i{}_j, d_2{}^i{}_j, \ldots, d_n{}^i{}_j)$$

The address of data item $d_k{}^i{}_j$ is determined by its relative position and the length of all data items before it. These lengths can themselves be other data items or part of the logical description of this physical record. The order of data items within a physical record is then important for finding their addresses. (A physical record here is "continuous" by definition.) The physical data base on level i is the set of physical records on this level

$$PB(i) = \{PR^i_1, PR^i_2, \ldots, PR^i_m\}$$

In reality only part of a physical data base on some level will exist at any time (Except the fourth level—the storage level—in which the whole physical data base exists.)

The definition of physical records is very simple because we believe that the complex structure of the data base is connected to the interpretation we give to these data items. Most of this interpretation is in the logical levels of the data base. In order to describe these logical levels, we need to define more concepts.

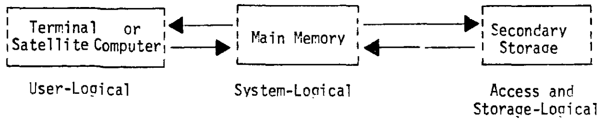


Figure 1—The four levels of a data base

Two data items are called *similar* if they have the same interpretation. A *field* is an abstract concept representing a set of similar data items. A field has no value but usually has a unique name or identifier. We denote field j on level i as F^i_j.

The notation $d_j \sim F_k$ means d_j is a *data item occurrence* of the field F_k. A *logical record* is a set of fields with a unique name or identifier. The order of fields in a logical record is immaterial because each field can be identified by its name. Logical records on level i are denoted as: LR^i_j.

What is the connection between logical records and physical records? Very simple. A physical record is an occurrence of a logical record and a data item is an occurrence of a field!

The logical data base on level i is a set of logical records plus their interpretation which is contained in the corresponding description language DL(i). The logical data base on level i is LDB(i).

$$LDB(i) = \{LR^i_1, LR^i_2, \ldots, LR^i_l\} + DL(i) + \text{some implicit interpretation.}$$

The connection between logical data base level, physical data base level, logical records and physical records is seen in Figure 2.

In order to better understand the concepts above we give two examples for describing structures which are common in data bases.

Example 1: *Repeating Group*

Suppose we have a repeating group which gives information on a person's children. In this example we disregard the level notation. The logical record is:

$$LRC = (F_1, F_2, F_3)$$

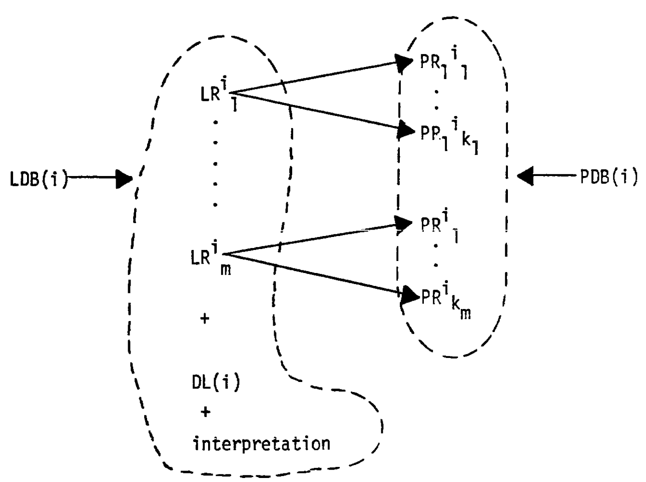


Figure 2—The connection between logical data base level, physical data base level, logical records and physical records

The following interpretation will appear as part of the logical description

LRC—information on children
F_1 —number of children
F_2 —name
F_3 —age

Physical records which are occurrences of this logical record are:

$$PR_1 = \{d_1, d_2, d_3, d_4, d_5\}$$

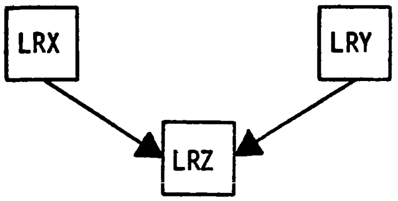
where $d_1 \sim F_1$, $d_2 \sim F_2$, $d_4 \sim F_2$, $d_3 \sim F_3$, $d_5 \sim F_3$ and value $(d_1) = V(d_1) = 2$, $V(d_2) =$ 'AMI', $V(d_3) = 25$, $V(d_4) =$ 'ROMIT', $V(d_5) = 22$ and

$$PR_2 = \{d_1, d_2, d_3, d_4, d_5, d_6, d_7\}$$

where $d_1 \sim F_1$; d_2, d_4, $d_6 \sim F_2$; d_3, d_5, $d_7 \sim F_3$ and $V(d_1) = 3$, $V(d_2) =$ 'SMITH', $V(d_3) = 29$ etc.

Example 2: *Sets*

Suppose we want to describe the structure:



LRX, LRY, and LRX are names for logical records.

$$LRX = \{F^x_1, F^x_2, \ldots, F^x_m, F^x_{m+1}\}$$
$$LRY = \{F_1^y, F_2^y, \ldots, F_n^y, F_{n+1}^y\}$$
$$LRZ = \{F_1^z, F_2^z, \ldots, F_k^z, F_{k+1}^z, F_{k+2}^z\}$$

with the following interpretation

F^x_{m+1}—address of the physical record occurrence LRZ which is the "son" (if there is more than one son, we need another field to specify the number of sons).

F_{m+1}^y—address of son LRZ
F_{k+1}^z—address of LRX parent
F_{k+2}^z—address of LRY parent

We do not show here physical record occurrences of the above logical records. It should be clear now that any structure whether inter or intra record can be represented in this way. The important fact is that the complex structure is part of the logical description while the physical records are no more than ordered tuples of data items.

The user logical level is denoted as U_1 (DB), U_2 (DB), ..., U_k (DB). We denote logical records on Level 1 as: UR_{ij}. Therefore

$$U_i(DB) = \{UR_{i1}, UR_{i2}, \ldots, UR_{in_i}\}$$

We denote a logical record on Level 2 as LR_j. The system logical level is denoted as:

$$S(DB) = \{LR_1, LR_2, \ldots, LR_s\}$$

We do not put any constraints on the transformations between these two levels. For example in the CODASYL model we have the following constraint:

$$\forall UR_{ij}\ \exists LR_k \quad \text{s.t. } UR_{ij} \subseteq LR_k (*)$$

We will allow more complex transformations. In the access level we introduce concept of *access record.* An access record is simply another name for a logical record on Level 3. We denote access record i as AR_i. The access level is then,

$$A(DB) = \{AR_1, AR_2, \ldots, AR_m\}$$

The transformation between the system logical level and the access level is very important and we would like to give some examples of it:

Example 3:

Suppose we have a file—$LR = \{F_1, F_2, \ldots, F_n\}$—and *two indexes.* The first index specifies for each data item occurrence of field F_1 the record(s) which contain this data item. The second index does the same for field F_2. We say that the logical record LR is translated into three *access records:*

$$AR1 = (F_{11}, F_{12})$$
$$AR2 = (F_{21}, F_{22})$$
$$AR3 = (F_{31}, F_{32}, \ldots, F_{3n})$$

where we have the interpretation $F_{11} = F_1, F_{12} =$ address of record containing a specific data item occurrence of F_1. (If there is more than one such record we need another field to specify the number of these records.)

$F_{21} = F_2$, $F_{22} =$ address of record containing a specific data item occurrence of F_2.

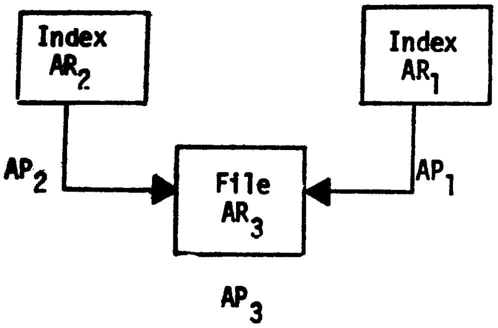
$$F_{31} = F_1,\ F_{32} = F_2, \ldots, F_{3n} = F_n$$

We therefore have three *access paths* to physical occurrences of LR:

$AP_1 =$ access path $1 = (AR_1, AR_3)$

$AP_2 = (AR_2, AR_3)$

$AP_3 = (AR_3)$ which represents a sequential search through the file. The situation is shown below:



With the concepts of access records and access paths, we can present now any structure of directories or indexes.

Example 4—*Inverted File*

$$LR = (F_1, F_2, \ldots, F_n) \text{ (This is also } AR_2)$$
$$AR_1 = (F_{11}, F_{12}, F_{13}, F_{14}) \text{ (directory entry)}$$

with the following interpretation:

$$\text{keyword} = \begin{cases} F_{11} = \text{name of field } F_i \\ F_{12} = \text{value of a data item occurrence on field } F_i \end{cases}$$
$$F_{13} = \text{number of records containing the corresponding keyword.}$$
$$F_{14} = \text{addresses of these records}$$

Notice that F_{13}, F_{14} represent together a repeating group. The possible access paths are

$AP_1 = (AR_1, AR_2)$—using the directory

$AP_2 = (AR_2)$—a sequential search

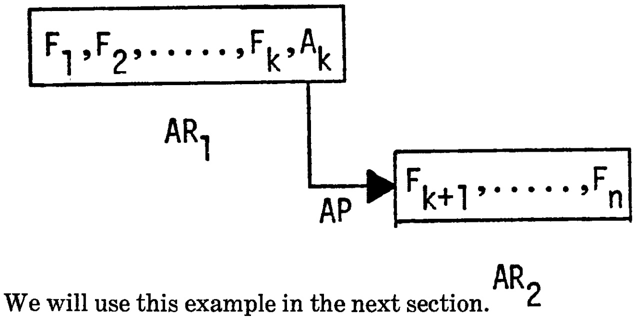
Example 5: *Splitting*

$$LR = (F_1, F_2, \ldots, F_n)$$
$$AR_1 = (F_1, F_2, \ldots, F_k, A_k)$$
$$AR_2 = (F_{k+1}, F_{k+2}, \ldots F_n)$$

A_k is the address of the occurrence of the second half of the same logical record. The *only possible* access path is:

$$AP = (AR_1, AR_2)$$

This situation is shown below:



We will use this example in the next section.

We feel that the concepts of access records and access paths are very significant since they clarify much of the confusion about the question: to which level do indexes and directories belong? We denote the logical records in Level 4 (storage level) as LP_i. The storage level is then:

$$P(DB) = \{LP_1, LP_2, \ldots, LP_p\}$$

In many cases this level is equivalent to the access level, i.e. $LP_i = AR_j$. However it may differ in two important aspects:

1. Control information used by the device such as "gaps" or keys may be added.
2. If cryptographic transformations are used on

(*) The notation $\subseteq$ here is used more by its semantic meaning rather than by its exact mathematical meaning.

this level, they may destroy the whole data item structure. In that case we can speak on another level—Level 5—called *unstructured storage level.*

Until now we discussed mainly the logical abstract levels. As was explained before, the physical records are just occurrences of the corresponding fields. One comment is appropriate at this point. It may be that in reality we will have a smaller number of physical media, and therefore of physical levels, than the number of logical levels. In that case, some of the logical levels are abstract and no corresponding physical level exists. The importance of having these abstract intermediate logical levels is for understanding the structure. For example, it is usually the case that the access level and the storage level correspond to one physical medium—the secondary storage. In this case, the physical records on Level 4 are occurrences of the logical records on Level 4. No physical records on Level 3 exist. The existence of the access records is important because otherwise the transformation $LR_i \rightarrow LP_j$ would be very difficult to understand. *In the case that we have only one physical medium, we come to the traditional view of a data base of several logical levels and one physical level!* So our model contains the traditional view as a special case.

CRYPTOGRAPHIC TRANSFORMATIONS

Now, with the notation defined, we can return to the subject of cryptographic transformations. These transformations can affect both the structure of the data and its coding or representation. The importance of having several physical levels for a data base should now be clear. We can look at the cryptographic transformations as transformations between two consecutive physical levels. *In general, cryptographic transformations are a subset of the set of transformations between the physical levels of the data base, which is used for protection.* Cryptographic transformations can be a powerful protection mechanism. They can be used in two ways:

(1) As part of the standard access control. That is, serving as a protection mechanism for implementing the protection specifications *as they are defined in each level.* In this case the user must know about the existence of these transformations since he probably holds some of the cryptographic keys.

(2) As a *system tool* for protecting against illegal access paths such as: browsing or wire tapping. In this case the cryptographic transformations are completely controlled by the system, they are not connected to a specific protection specification and the user does not have to know about their existence.

Of course, the combination of these two methods can be used. We leave the subject of how to use cryptographic transformations to the next section and concentrate in this section on describing the possible cryptographic transformations between the different physical levels of the data base.

TRANSFORMATIONS BETWEEN PHYSICAL LEVELS

We start with the transformation between the user-logical and the system-logical levels. Clearly, the transformation between the corresponding physical levels is dependent on the transformation between the logical levels. In general, these transformations can be very complex. However, if we want to use system services to process and query on data items and we want the ability of sharing data with other users, the common data item structure has to be preserved. We identify three types of transformations between physical Levels 1 and 2:

Type 1: *Data Item Substitution*

$$d^1_i = f(d^2 j)$$
$$d^2_j = f^{-1}(d^1 i)$$

This is the common substitution transformation which does not destroy the data item structure. Its advantage is its simplicity and the fact that some processing, such as querying, can be done on data items even in their ciphered form. Its disadvantage is that it may not be secure enough. (Because the statistical properties of similar data items may be saved.)

Type 2: *Data Item Expansion*

$$d^2_i = f(d^1_j, d^1_{j+1}, \ldots, d^1_{j+k})$$

A data item in the system logical is expanded to several data items and finer structure on the user logical level. As an example suppose we have the fields: NAME, AGE, SEX on the user logical level, while on the system logical level we have one field: PERSONAL DATA, whose occurrences are a scrambled form of occurrences of the three fields on the user logical level. This transformation can be made more secure than the former one but it has a major disadvantage which is: because the fine structure on the user logical is destroyed a user cannot ask queries, which require processing on the system logical level, about individual fields such as: NAME, AGE, SEX since these fields are not defined in the system logical level.

A third type of transformation is *data item contraction*

$$d^1_i = f(d^2_j, d^2_{j+1}, \ldots, d^2_{j+k})$$

Clearly more complex transformations can be defined between Level 1 and 2. One comment is appropriate at this point. There may be some cryptographic transformations between the two levels that we are not aware of. For example, if Level 1 and Level 2 are far apart and are connected by telephone lines, some enciphering device may be connected to these lines. We are not interested in this kind of enciphering which has nothing to do with the data base structure and as far as the representation of the data in the two levels is concerned does not have any effect. The same comment is true in the case of transformations between other physical levels of the data base.

System Logical⟵- - -⟶Access⟵- - -⟶Storage

We discuss the transformations between these three levels together since we consider the case of one physical medium to both Levels 3 and 4. We identify three types of transformations:

Type 1: *Substitution Oriented*

$$d^2_i=f(d^4_j),\ d^4_j=f^{-1}(d^2_i)$$

The importance of this transformation is that it allows us to define *processable ciphers.* Most of the processing of data items is done on the system logical level (in main memory). So our definition of processable ciphers must be connected to data in Level 2. The advantages of being able to process data in its ciphered form were discussed above. Informally, a processable cipher is a transformation from the system logical level to the storage level which preserves the data item structure and allows the type of operations which are done on the "clear" data to be done on the ciphered data items. Formally, if $d^2_1=f(d^4_1)$ and $d^2_2=f(d^4_2)$ and G is a defined operation on data items in Level 2, then f is processable if and only if:

$$\exists\, G,G' \rightarrow G(d^2_1,d^2_2)=f(G^1(d^4_1,d^4_2))$$

Usually we also require that $G=G^1$. For example, if G is the *compare* operation, then we have

$$d_1{}^4=d_2{}^4 \Leftrightarrow f(d_1{}^4)=f(d_2{}^4) \Leftrightarrow d_1{}^2=d_2{}^2$$

Such a cipher f is called a *retrievable* cipher, since it allows the searching and retrieving of records in their ciphered form. (This is done by comparing the ciphered form of the query to the ciphered content of the record.)

Type 2: *Transposition Oriented*

We have explained before that the order of data items in a physical record is important in order to find their addresses. However, this order may change from one physical record to another. For example, suppose a logical record on Level 4 is: $LP=(F_1,F_2,F_3)$ and following are occurrences of physical records:

$$PR_1=(d_{11},d_{12},d_{13})$$
$$PR_2=(d_{21},d_{22},d_{23})$$
$$PR_3=(d_{31},d_{32},d_{33})$$

Where:

$$\begin{array}{lll} d_{11}\sim F_1, & d_{22}\sim F_1, & d_{33}\sim F_1 \\ d_{12}\sim F_2, & d_{21}\sim F_2, & d_{32}\sim F_2 \\ d_{13}\sim F_3, & d_{23}\sim F_3, & d_{31}\sim F_3 \end{array}$$

That is, the order of data items in a physical record changes from one record to another. Such a transformation is very effective against "browsing" since the starting address of data items is not known to the illegal "browser". Its disadvantages are that it increases the access time of finding a data item, and that another data item has to be added which contains information about the specific order in each record.

Type 3: *Access Oriented*

In this transformation we encipher only the data items which allow the transfer from one access record to another in a specific access path. For example, suppose we use the *splitting* example above. We have

$$LR=(\text{NAME}, F_2, \text{SALARY}, F_4)$$

and

$$AR_1=(\text{NAME}, F_2, A)$$
$$AR_2=(\text{SALARY}, F_4)$$

A—is the address of the second half of an occurrence of LR. If we encipher the field A, then matching the right salary to the right name without deciphering this field is very difficult. The reason is that we have used the notion of *access path* for enciphering. In this case we enciphered the *only existing* access path. This type of transformation can be very effective, since address fields do not have regular statistical properties. (Note also that fields NAME and SALARY do not have to be enciphered.)

COMBINED TRANSFORMATIONS

Shannon[15] has shown the strength of combined cryptographic transformations. The simplest example of a combined transformation is the substitution—transposition transformation. In this case data items are first enciphered by substitution transformation and then transposed by the transposition transformation. It is very difficult now to get the statistical properties of data items in order to "break" the substitution because of the transposition.

Another effective combination is the access-substitution combination. Following are two examples of substitution access combination.

Example 1: *Hashing*

Suppose we have a hashing "file". We have one logical record

$LR=(F_1,F_2,\ldots,F_n)$ and two access records

$AR_1=(F_1)$—F_1 is the hashing field

$AR_2=(F_1,F_2,\ldots,F_n)$

We have two access paths. The legal access path—(AR$_1$,AR$_2$) and the illegal one—(AR$_2$) which is equivalent to a sequential search. Suppose we use the following transformation:

$$k+d \rightarrow \textit{hashing address}+K^1$$

where d$\sim$F$_1$ and K is the cryptographic key. The meaning of this transformation is that as a result of the hashing process we get an address and a key K^1. Key K^1 which is the "residue" of the hashing transformation is used for the substitution transformation of data item d. The illegal path of (AR$_2$) is now very difficult, since in order to guess what the substitution is, we need to go through the legal access path (AR$_1$, AR$_2$) using the hashing algorithm and the cryptographic key K!

Example 2: *Inverted File*

Suppose we have the "file" described above in Example 4. That is

$$LR=(F_1,F_2,\ldots,F_n)$$
$$AR_1=(F_{11},F_{12},F_{13},F_{14})$$
$$AR_2=(F_1,F_2,\ldots,F_n)$$

The two possible access paths are (AR$_1$,AR$_2$) and (AR$_2$). We add to access record AR$_1$ a KEY field. That is

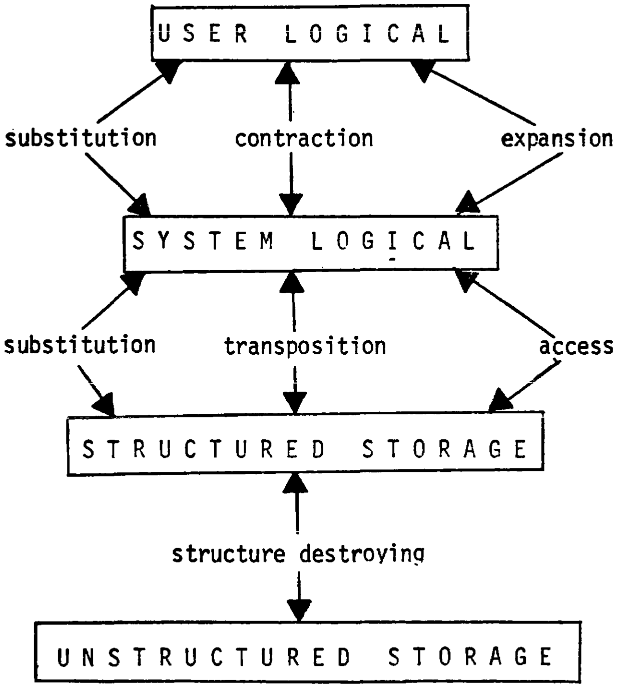
$$AR_1=(F_{11},F_{12},F_{13},F_{14},F_{15})$$

where F$_{15}$ is the key to encipher records with a specific keyword. All records which contain a data item occurrence of Field F$_1$ (=keyword) are enciphered by specific occurrence of field F$_{15}$. Therefore all records which have the same keyword are enciphered by the same substitution key. Again illegal path (AR$_2$) is protected by the fact that for deciphering we need to know which records contain a particular keyword, but this information is contained in the directory only, which is part of the legal access path. (*) Many variations of the combined transformation technique are possible. One possible extension of this is to build the directory as a *tree* where the key for enciphering a node of the tree is contained in its parent node. The subject of the enciphering of search trees is discussed also in Bayer and Metzger.[2]

Storage←- - - - - - →Unstructured Storage

One can think of a general structure destroying transformation which can be used, for example, for backup or data migration purposes. Since usually another physical medium is involved (e.g., magnetic tape) we define another logical and corresponding physical level called: *unstructured storage level.* We do not give any special notation for this level. (It should be straightforward using the examples above.) An example for such transformation is to use a pseudo-random number generator to encipher a complete "file" or a large part of a data base. Such transformation can be made very secure. Its major disadvantage is that complete deciphering is necessary before any access is possible.

We summarize this section with the following schematic diagram of types of cryptographic transformations and their use in the multi-level data base model:



USING CRYPTOGRAPHIC TRANSFORMATIONS

As was pointed out above, there are two ways for implementing cryptographic transformations: (1) as part of the standard protection mechanism according to some protection specifications. These transformations are user oriented (since the protection specifications are user oriented) and the users should know about their existence, since they probably have some of the cryptographic keys, (2) as a system tool to protect against illegal access paths such as "browsing" or "wiretapping". In this case users may not be aware of the existence of these transformations and their keys are under system (or DBA) control.

The actual implementation of the two methods depends on the data base structure and on the way the protection specifications and mechanism are spread out through the different levels of the data base.

(*) The case where a record contains two keywords or can be accessed by more than one access path can be further researched based on McCauley.[12]

SYSTEM CONTROLLED TRANSFORMATIONS

The following transformations are examples for system controlled transformations (method 2 above)

(1) The transformation between the unstructured storage and the structured storage levels are always system controlled.
(2) Combined transformations such as substitution-transposition or substitution—access can be system controlled for protecting against the illegal access path or browsing (e.g., in case of "stealing" a physical secondary storage device). In this case the cryptographic keys for these transformations have to be *internal* to the data management or access control methods.
(3) Substitution oriented transformations, including processable ciphers under system control, can be used to protect against wiretapping or against dumping of main memory. Again, the keys for these transformations are either internal to the processing primitives of the query language or are part of the DDL for Level 2, but they must be protected by standard access control methods.

We see then that system controlled transformations are easy to implement and quite simple to control. They are controlled by the DBA and he can change the cryptographic keys at any time. They are very effective against illegal access paths such as: "browsing" or "wiretapping" but they must be complemented by standard access control methods for protection of the cryptographic keys. This method, of using both system controlled cryptographic transformations and standard protection mechanisms (e.g., passwords) such that each mechanism protects the "weak" areas of the other, can be very effective provided that adequate administrative security exists so that the probability of compromising both mechanisms is low.

USER CONTROLLED TRANSFORMATIONS

The situation with user controlled transformations is more complex, and the main reason is the existence of sharing and overlapping protection specifications. Clearly, if every user has access to a unique part of the data base, then this part can be enciphered by a unique key and only the appropriate user has the right key. But this case is unreal since the main purpose of a data base is *sharing data.* McCauley[12] has suggested a way to *partition* a data base using *security atoms.* The security atoms are data base dependent and not protection specification dependent! Every user has access to a set of security atoms according to the protection specifications. A way to use cryptography in this model will be to encipher each security atom with a unique key and distribute the keys between users according to the protection specification. (Somewhat similar to the passwords mechanism.)

Another possible use of user controlled cryptography is based upon the fact that the only "clear" data which is sent to the user is the data that he should have legal access to. (The fact that he gets a lot of "garbage" may discourage him from issuing illegal queries.) The main advantage of user controlled transformations versus system controlled transformations is concerned with the problem of *key placement.* In the system controlled case, the cryptographic keys must reside at all times in the system and therefore must be protected by the standard protection mechanisms. In the user controlled case there is the possibility that only the user(s) will have some of the cryptographic keys to their authorized data. These keys do not reside in the system except at the actual time the user is accessing his data. In this case even the DBA cannot "decipher" the user's data since he does not have the right key. This is certainly in accordance with the "need to know" concept, i.e., that the DBA does not have to know the content of every data item in the data base. Such a system, where only the authorized user has the cryptographic key was implemented by IBM[4] using the LUCIFER system to protect a "secure field." There are two disadvantages to this method. The first is the possibility of penetration of illegal cryptographic keys. Since the right key is not stored in the system, some checking algorithm (e.g., check digit) must be used. The probability that an illegal key will not be detected by the checking algorithm is not zero and its results in case of updating can be disastrous! The second disadvantage is that it is very hard to control the "user controlled transformations" case. For example, close cooperation between users which share the same data is needed if the cryptographic keys are to be changed. Also, close cooperation is needed between users and the DBA in some cases of error recovery and data base reorganization.

We note that cryptographic transformations which are controlled by the system are easier to implement but since the keys must reside within the system, they may be less secure. User controlled transformations allow the possibility that only the user will have the right cryptographic key, and therefore provide more security, but they require data base partitioning and probably a large number of keys, and therefore present quite a serious control problem.

FURTHER RESEARCH

The model presented in this paper is used as a *framework* for further research in the area of cryptography and data base security. This research is reported in Gudes.[8] The main areas of this further research are: (1) a detailed investigation into the problem of how to design an authorization scheme which is based on user controlled cryptographic transformations. Several types of protection specifications such as: compartmentalized, hierarchical or data dependent are discussed and several cryptographic schemes are suggested for their implementation, (2) research into the problem of *evaluating* the security of data base enciphering. A "security measure" for data base enciphering is suggested and is computed for several types of ciphers, (3) an experiment using a simple file system is being performed. The goal of this experiment is to gain insight on the amount of security provided by the different ciphers and the overhead associated with them. The results of these areas of the research will be reported in future papers.

SUMMARY

In this paper we have investigated the problem of applying cryptographic transformations for enhancing the security of data base systems. These transformations have been recognized in the past as a valuable protection mechanism but their relation to data base security has not been identified. The major reason was the lack of a suitable data base model with which the problems related to security and cryptography can be analyzed. Such a model is developed in this paper. This is a multi-level model of a data base which helps us to understand the connection between the data base structure and the cryptographic transformations applied to the data base. We have identified several general types of cryptographic transformations, and have shown the two major ways in which they can be used: as system controlled transformations, or as user controlled transformations. More research is being conducted on the problem of designing a secure system based on cryptography and on the cost-effectiveness of the application of cryptographic transformations in a data base system.

REFERENCES

1. Astrahan, M. M., E. B. Altman, P. L. Fehder and M. E. Senko, "Concepts of a Data Independent Accessing Model," *SIGFIDET Workshop*, 1972.
2. Bayer, B. and J. K. Metzger, "On the Encipherment of Search Trees and Random Access Files," *Proceedings of First International Conference on Very Large Data Bases*, September, 1975.
3. *CODASYL Data Base Task Group Report*, April, 1971.
4. Feistel, H., W. Notz and L. J. Smith, "Some Cryptographic Techniques for Machine to Machine Data Communications," *Proceedings of the IEEE*. November, 1975.
5. Friedman, T. D., "The Authorization Problem in Shared Files," *IBM System Journal*, Vol. 7, No. 4, 1970.
6. Graham, R. M., "Protection in an Information Processing Utility," *CACM*, Vol. 15, No. 5, May, 1968.
7. Graham, S. G. and P. J. Denning, "Protection Principles and Practice," *AFIPS Conference Proceedings*, SJCC, 1972.
8. Gudes, E., *The Application of Cryptography for Data Base Security*, Ph.D. Dissertation, The Ohio State University, 1976 (to appear).
9. Hoffman, L. J., "The Formulary Model for Flexible Privacy and Access Controls," *AFIPS Conference Proceedings*, FJCC, 1971.
10. IBM Systems Reference Library, *OS Data Management Services Guide*, Order No. GC26-3746, 1972.
11. Lampson, B. W., "Dynamic Protection Structures," *AFIPS Conference Proceedings*, FJCC, 1969.
12. McCauley, E. J., *A Model for Data Secure Systems*, Ph.D. Dissertation, The Ohio State University, 1975.
13. Minsky, N., "On Interaction with Data Bases," *ACM-SIGFIDET Workshop on Data Description, Access and Control*, May, 1974.
14. Petersen, H. E. and R. Turn, "Systems Implications of Information Privacy," *AFIPS Conference Proceedings*, SJCC, 1967.
15. Shannon, C. E., "Communication Theory of Secrecy Systems," *The Bell System Technical Journal*, Vol. 28, No. 4, 1949.
16. Sibley, E. H., "On the Equivalence of Data Base Systems," *SIGFIDET Workshop on Data Description, Access and Control*, May, 1974.
17. Sibley, E. H. and R. W. Taylor, "A Data Definition and Mapping Language," *CACM*, Vol. 16, No. 12, December, 1973.
18. Skatrud, R. D., "A Consideration of the Application of Cryptographic Techniques to Data Processing," *AFIPS Conference Proceedings*, FJCC, 1969.
19. Turn, R., "Privacy Transformations for Databank Systems," *NCC Proceedings*, 1973.
20. Van Tassel, D. L., "Cryptographic Techniques for Computers," *AFIPS Conference Proceedings*, SJCC, 1969.

Multiuser cryptographic techniques*

by WHITFIELD DIFFIE and MARTIN E. HELLMAN
Stanford University
Stanford, California

ABSTRACT

This paper deals with new problems which arise in the application of cryptography to computer communication systems with large numbers of users. Foremost among these is the key distribution problem. We suggest two techniques for dealing with this problem. The first employs current technology and requires subversion of several separate key distribution nodes to compromise the system's security. Its disadvantage is a high overhead for single message connections. The second technique is still in the conceptual phase, but promises to eliminate completely the need for a secure key distribution channel, by making the sender's keying information public. It is also shown how such a public key cryptosystem would allow the development of an authentication system which generates an unforgeable, message dependent digital signature.

INTRODUCTION

In a computer network with a large number of users, cryptography is often essential for protecting stored or transmitted data. While this application closely resembles the age old use of cryptography to protect military and diplomatic communications, there are several important differences which require new protocols and new types of cryptosystems. This paper addresses the multiuser aspect of computer networks and presents ways to preserve privacy of communication despite the large number of user connections which are possible.

In a system with n users there are n^2-n pairs who may wish to hold private conversations. The straightforward way to achieve this is to give each pair of users a key in common which they share with no one else. Each user will then have n-1 keys, one for communicating with each other user. Unfortunately, the cost of distributing these keys is prohibitive. A new user must send keys to all other users. Unfortunately, the network cannot be used for this purpose, and an external secure channel is required. This procedure is comparable to requiring each new telephone subscriber to send a registered letter to everyone else in the phonebook.

Military communications suffer less from this problem for several reasons. Among these are the limitations imposed by the chain of command and the fact that stations change allegiance infrequently. In a computer network designed for business communication, on the other hand, users will regard each other as friends on one matter and as opponents on another. Firms A and B may cooperate on one venture in competition with C, while simultaneously, A and C compete with B on a different endeavor. A must therefore use different keys for communicating with B and C.

One approach to this problem is to assume that the users trust the network. Each user remembers only one key which is used to communicate with a local node. From there the message is relayed from node to node, each of which decrypts it, then reencrypts it in a different key for the next leg of its journey. This process is known as link encryption.[1] When the message reaches the network node closest to its destination, it is sent on to the addressee encrypted in a key shared only by the addressee and that node.

Although this technique requires each user to remember only one key, it has the disadvantage that a message is compromised if any one of the nodes in its path is subverted. In this paper we examine two other ways of allowing secure communication between any pair of users without assuming the integrity of all nodes in the network and without requiring the users to distribute or store large numbers of keys.

The first technique requires no new technology, but imposes a complex initial connection protocol. This is the subject of the second section of this paper. We call the second technique public key cryptography, since most of the secrecy traditionally required for the keys has been removed. This is discussed in section three and represents a radical departure from past cryptographic practices. While it requires further work before it becomes implementable, its simplicity of operation makes it extremely attractive. If a suc-

* This work was supported by the National Science Foundation under NSF Grant ENG 10173.

cessful implementation can be developed it should find wide use in both military and civilian applications.

The fourth section shows how public key cryptography can be used to provide a time and message dependent digital signature which cannot be forged even when past signatures have been seen. This is an example of the general problem of authentication discussed in greater detail in Reference 2, which provides a more general perspective in which public key cryptography can be viewed.

A PROTFCTIVE PROTOCOL

As indicated earlier, a message protected by link encryption will be compromised if any node in the path it follows from the sender to the receiver is subverted. In this section we describe a protocol which guarantees to protect the message unless a large number of nodes are compromised. While many variations are possible, the basic technique is as follows.

A small number m of the network's nodes will function as "key distribution nodes." Each user has m keys, one for communicating with each of these m nodes. These keys vary from user to user, so while each user must remember only m keys, each of the key distribution nodes remembers n, one for each user of the net. When users A and B wish to establish a secure connection they contact the m key distribution nodes and receive one randomly chosen key from each. These keys are sent in encrypted form using the keys which the users share with the respective nodes. Upon receiving these keys, the conversants each compute the exclusive or of the m keys received to obtain a single key which is then used to secure a private conversation. None of the nodes involved can violate this privacy individually. Only if all m nodes are compromised will the security of this connection fail.

It might be objected that any key distribution node acting alone can prevent all communication by mischievously sending out different keys to each of the parties, thus bringing network operations to a halt. The users, however, can easily protect themselves against this threat. If communication using the composite key fails, its use as a key is abandoned, and the components are exchanged one by one, in clear, for comparison. If any key fails to agree, the node which issued it is blacklisted. Finally, on conclusion of this process, the users repeat the request for keys to the nodes which passed the previous test.

Alternatively, the component keys can be compared by the use of one way functions[2,3,5] without ever being transmitted in clear. Loosely speaking, a function f is called a one-way function if it is easy to compute in the forward direction, but given any output, it is computationally infeasible to find an input which produces it. In referring to a task as computationally infeasible, we have in mind that it cannot be done in fewer than a finite but astronomical number of operations, say 2↑100. For practical purposes, this is equivalent to being incomputable. As shown in Reference 2, a one way function can easily be obtained from a secure cryptosystem.

If communication fails using the composite key, the users send the images of the individual keys under a public one-way function. If the image received does not agree with that computed by applying f to the key, the node which issued it is guilty of compromise. Since the valid keys have not been publicly revealed in this process, there is no need to request new ones from the uncompromised nodes. Instead the invalid ones are omitted and the remainder xored.

To sum up, this technique requires each user to remember m keys and each key distribution node to remember n keys. Unless all m key distribution nodes are subverted, any two users can establish a private link through use of a set-up protocol usually requiring 2m exchanges (more are required if a key distribution node has been subverted). The next section describes a concept which eliminates much of this overhead and does not require the user to trust any node. This new concept, if successfully implemented, will make the technique described above obsolete.

PUBLIC KEY CRYPTOGRAPHY

In this section we propose that it is possible to eliminate most of the secrecy surrounding the key used in a communication, and yet to preserve the secrecy of the communication. This is accomplished by giving each user a pair of keys E and D. E is an enciphering key and is public information. D is the corresponding deciphering key, and while this must be kept secret, it need never be communicated, eliminating the need for a secure key distribution channel. Although D is determined by E, it is infeasible to compute D from E.

For reasons of security, generation of this E-D pair is best done at the user's terminal which is assumed to have some computational power. The user then keeps the deciphering key D secret but makes the enciphering key E public by placing it in a central file along with his name and address. Anyone can then encrypt a message and send it to the user, but only the intended receiver can decipher it. Public key cryptosystems can therefore be regarded as multiple access ciphers.

By regularly checking the file of enciphering keys the user can guard against any attempt to alter it surreptitiously. Any such mischief is reported and settled by other authentication means, such as personal appearance.

The crucial feature of a public key system is that it is relatively easy to generate an E-D pair, preferably automatically through a publicly available transformation from a random bit string to E-D, and yet it is computationally infeasible to compute D from E.

At present we have neither a proof that public key systems exist, nor a demonstration system. We hope to have a demonstration E-D pair in the near future, and expect that if the demonstration pair successfully resists attack then we will be able to design an algorithm for automatically generating E-D pairs of a similar kind. In the meantime, the following reasoning is given to help dispel any doubts the reader may have.

A suggestive example is to let the cryptogram, represented as a binary n-vector c equal E m; where m is the message also represented as a binary n-vector, and E is an arbitrary n-by-n invertible matrix. Letting D=E↑-1 we have m=D c. Thus both enciphering and deciphering are easily accomplished with about n↑2 operations. Calculation of D from E, however, involves a matrix inversion which is a harder problem. And it is at least conceptually simpler to obtain an arbitrary pair of inverse matrices than it is to invert a given matrix. Start with the identity matrix I and do elementary row and column operations to obtain an arbitrary invertible matrix E. Then starting with I do the inverses of these same elementary operations in reverse order, to obtain D=E↑-1. The sequence of elementary operations could easily be generated from a random bit string.

Unfortunately, matrix inversion takes only about n↑3 operations even without knowledge of the sequence of elementary operations. The ratio of "cryptanalytic" time (i.e., computing D from E) to enciphering or deciphering time is thus at most n. To obtain ratios of 10↑6 or greater would thus require enormous block sizes. Also, it does not appear that knowledge of the elementary operations used to obtain E from I greatly reduces the time for computing D. And, since there is no round-off error in binary arithmetic numerical stability is of no consequence in the matrix inversion. In spite of its lack of practical utility, this matrix oriented example is still useful for clarifying the relationships necessary in a public key system.

A more practical direction uses the observation that we are really seeking a pair of easily computed inverse algorithms E and D, but that D must be hard to infer from E. This is not as impossible as it may sound. Anyone who has tried to determine what operation is accomplished by someone else's machine language program knows that E itself (i.e., what E does) can be hard to infer from E (i.e., a listing of E). If the program were to be made purposefully confusing through addition of unneeded variables, statements and outputs, then determining an inverse algorithm could be made very difficult indeed. Of course, E must be complicated enough to prevent its identification from input-output pairs.

Another idea appears more promising. Suppose we start with a schematic of a 100 bit input, 100 bit output circuit which merely is a set of 100 wires implementing the identity mapping. Select 4 points in the circuit at random, break these wires, and insert AND, OR and NOT gates which implement a randomly chosen 4 bit to 4 bit invertible mapping (a 4 bit S box in Feistel's notation).[4] Then repeat this insertion operation approximately 100 times to obtain an enciphering circuit E. Knowing the sequence of operations which led to the final E circuit allows one to easily design an inverse circuit D. If however the gates are now randomly moved around on the schematic of E to hide their associations into S boxes, an opponent would have great difficulty in reconstructing the simple description of E in terms of S boxes, and therefore would have great difficulty in constructing a simple version of D. His task could be further complicated by using reduction techniques (e.g. Carnaugh maps) or expansion techniques (e.g. $\sim(AB) = \sim A$ or $\sim B$, or expressing a logical variable in terms of previous variables), and by adding additional, unneeded S boxes and outputs.

For ease of exposition, we have described the implementation of a specific key in hardware. In practice, a special purpose simulator is obviously of most interest. The hardware description is also valuable in exemplifying a generally useful idea. To build a good public key cryptosystem one needs easily inverted elementary building blocks and a general framework for describing the concatenation of these elementary blocks. Here the elementary building blocks are S boxes and the general framework is the schematic diagram. The general framework must also hide the sequence of elementary building blocks so that no one other than the designer can easily implement the sequence of inverse elementary operations. Examination will show that the matrix example had a similar structure, except there the general class of transformations obtainable was too small.

While the above arguments only provide plausibility as opposed to proof, we hope they will stimulate additional work on this promising area of research.

PUBLIC KEY AUTHENTICATION

The purpose of a cryptographic system is to prevent the unauthorized extraction of information from a public (i.e., insecure) channel. The dual problem of authentication is to prevent unauthorized injection of messages into a public channel.

In conventional paper oriented business transactions, signatures provide a generally accepted level of authentication. As electronic communication replaces mail service the need for a digital signature will be strongly felt.

Various types of authentication are now possible,[2] but the development of public key cryptosystems would allow an entirely new dimension.

Currently, most message authentication consists of appending an authenticator pattern, known only to the transmitter and intended receiver, to each message and encrypting the combination. This protects against an eavesdropper being able to forge new, properly authenticated messages unless he has also stolen the key being used. There is no protection against such an eavesdropping thief or against the threat of dispute. That is, the transmitter may transmit a properly authenticated message, later deny this action, and falsely blame the receiver for taking unauthorized action. Or, conversely, the receiver may take unauthorized action, forge a message to itself and then falsely blame the transmitter for these actions. For example, a dishonest stockbroker may try to cover up unauthorized buying and selling for personal gain by forging orders from clients. Or a client may disclaim an order, actually authorized by him, but which is later seen to cause a loss. We will introduce concepts which would allow the receiver to easily verify the authenticity of a message, but which prevent him from generating apparently authenticated messages, thereby protecting against both the threat of eavesdropping thieves and the threat of dispute. Note that these techniques thus provide stronger protection than signatures, voiceprints, etc. which can be forged once seen and are not message dependent.

To obtain an unforgeable digital signature from a public key cryptosystem, the protocol would be as follows: Assume user A wishes to send a message M to user B. The transformed message C=EbDa(M) is sent, where Eb represents the transformation effected by use of B's public enciphering key and Da represents the transformation effected by use of A's secret deciphering key. Upon receipt of C, user B operates first with his secret operation Db and then with the public operation Ea thereby obtaining EaDb(C) =EaDbEbDa (M) =M. No one else can extract M because of the need to know Db. By saving the intermediate result Db(C) =Da(M) user B (and only user B) can prove that he received the specific message M from user A. There must be some structure to the message (e.g., it could include a date and time field) to prevent injection of random bit patterns for C, with the hope that the resultant decoded "message", EaDb(C), might cause random mischief such as deletion of files.

Note that since there is no need for a secure channel for distribution of authentication information, we have a public key authentication system. This system protects against, "eavesdropping thieves" and against a dispute as to whether or not an action taken by the receiver was authorized by the transmitter. Similarly, a public key cryptosystem can be used to protect against the other type of dispute in which the transmitter A claims to have issued an order which was not carried out by the receiver B. The transmitter requests that the receiver B send EaDb(M) as a receipt for the message M. By operating on this receipt with his secret operation Da, the transmitter obtains Db (M), which could only have been generated by the receiver B. Only user A can generate this receipt since it requires knowledge of Da.

While the above discussion centered on message authentication it also applies to user authentication. The implicit message becomes "I am user X and the time is T." Inclusion of the time field prevents an eavesdropper from using old authentication signals to pose as someone else. For reasons noted in Reference 2, such a system deserves to be called a one-way IFF system.

We thus see that public key cryptosystems developed for ensuring the privacy of communications, could also be used to ensure their authenticity. They could therefore be used to fill the need for a digital equivalent of a signature. This need is currently a major barrier to the use of electronic mail for business communications, and provides additional motivation for study of public key cryptosystems.

ACKNOWLEDGMENT

The authors wish to thank Leslie Lamport of Massachusetts Computer Associates for several valuable discussions. In particular, the technique described in Section 2 was discovered during one of these conversations.

REFERENCES

1. Baran, Paul, *On Distributed Communications: IX. Security, Secrecy, and Tamper-Free Considerations*, Santa Monica, CA: The Rand Corporation August 1964, (RM-3765-PR).
2. Diffie, Whitfield and Martin E. Hellman, forthcoming paper to be submitted to the *IEEE Transactions on Information Theory*.
3. Evans, Arthur, Jr., William Kantrowitz and Edwin Weiss, "A User Authentication System not Requiring Secrecy in the Computer," *Communications of the ACM*, Vol. 17 No. 8, August 1974, pp. 437-442.
4. Feistel, Horst, "Cryptography and Computer Privacy," *Scientific American*, Vol. 228, No. 5, May 1973, pp. 15-23.
5. Purdy, George B., "A High Security Log-in Procedure," *Communications of the ACM*, Vol. 17, No. 8, August 1974, pp. 442-445.

Cryptography using modular software elements*

by HERBERT S. BRIGHT and RICHARD L. ENISON
Computation Planning, Inc.
Bethesda, Maryland

ABSTRACT

Protection of information within a computer/communication system can be provided through reversible cryptographic transformation of the information itself into a form that can be returned to usable form only through use of control information known as "key."

It is not necessary, in order to achieve access control, that the encryption algorithms, random number generator, or system organization be kept secret; in fact, a basic requirement of modern cryptographic technology is that it must be effective although a would-be penetrator is assumed to have full access to all of that information and the facilities and competence to apply it. Only the key can be assumed to be, and must be, physically secure.

The building-block approach outlined makes use of pre-programmed software elements for providing all specialized algorithms, including the Proposed Federal Data Encryption Standard (DES), together with necessary nonnumeric generalized support routines for use with application programs written in conventional procedural higher languages (FORTRAN, COBOL, etc.). Both Strong Algorithm and Long Key methods can be used as required by security-level-vs-cost tradeoff considerations.

This method is useful in conjunction with specialized hardware; for testing of programs and hardware; in some cases instead of hardware; and can support multiple-level security applications.

The entire scheme, including the Tausworthe-Lewis-Payne bitwise linear recurrence modulo 2 quasirandom number generator, is based irrespective of hardware type on a standardized 64-bit data element.

INTRODUCTION

In spite of appropriate bars and locks on parts of a system: If the information it handles is in a form that can be understood, used, or damaged by unauthorized persons without such access being immediately evident to management, the information is vulnerable.

Loss of management control over sensitive information or operations is then prevented only by the integrity of system controls.

Unauthorized access may be for improper or malevolent purpose or—much more probably—may occur by accident; but in any case it is ultimately management's responsibility to avoid errors and omissions in planning that can lead to such vulnerability.

In the course of physical and logical configuration planning, management has some choices of "appropriate bars and locks" in hardware/software installation.

Protection of the information itself, through privacy transformation using cryptographic technology, provides an additional level of management control that can be relatively low in cost and high in effectiveness.

Elementary applications of cryptographic protection can include:

- User authentication (off-line)
- Terminal authentication (on-line)**
- Data link protection
- Network protection
- File access protection

The basic concept of modern cryptographic protection for information is that access to it can be limited to only those properly authorized to make such access, merely by protecting an information key.

This paper will outline a method by which preprogrammed software elements can be combined in modular fashion to provide a broad range of cryptographic transformation capabilities.

CRYPTOGRAPHY CONCEPTS

Cryptographic transformation of discrete data elements consists of applying deterministic modification processes that prevent the data from being recognized or used, or modified without such modification becom-

* This work was done in connection with software product development.

** An advanced two-way authentication scheme is outlined in Reference 1.

ing obvious. This paper will be limited to reversible crypto processes (i.e., those which are capable of subsequently reversing the encryption (i.e., "decrypting")) in order to recover the original text.

Basic ideas

Figure 1 defines the nomenclature and shows schematically the relationship between input data (known as Plaintext or Cleartext); the encrypted data (known as Cipher or Ciphertext); the data used to control encryption and decryption (known as Key); and the encrypt/decrypt operations.

A system using the arrangement of elements shown in Figure 1, constituting the simplest and most basic implementation of cryptography, is known by the term "Electronic Code Book" or ECB. Many writers distinguish between encoding (conversion of message elements to different form using a substitution process) and enciphering (transforming message elements using an algorithmic process). Enciphering transformations include both resequencing and in-sequence conversion.

Shannon, whose basic 1949 work[2] is fundamental to much subsequent work and is the foundation for the work reported in the present paper, distinguished between

- "Concealment Systems," also known as Steganography, in which the very existence of sensitive information is concealed;
- "Privacy Systems," in which the physical form of information is transformed and reconverted by special equipment assumed to be not possessed by unauthorized users; and
- " 'True' Secrecy Systems," in which the information is modified only logically, and unauthorized persons can be assumed to be aware of its existence and to have any equipment needed to decrypt it into Plaintext form.

We will consider only what Shannon called True Secrecy Systems: We assume that the algorithm(s) used for data transformation, and all necessary system information, are completely public and that only the key information is physically secure; viz., that a key is known only to authorized persons who are authorized to access the information that is to be protected by cryptographic transformation. This assumption is consistent with the National Bureau of Standards' Draft Guidelines[3] in which D. K. Branstad established the basic assumptions underlying development and use of the Proposed (U.S.) Federal Data Encryption Standard (DES).

Shannon showed that effective cryptographic transformation processes can be implemented using either algorithmic complexity or key length for achieving strength: He showed that what he called "product systems," consisting of a combination of simpler cryptographic processes, form a linear associative algebra with a unit element, thus permitting techniques to be concatenated without losing the required deterministic nature of the resulting process.

The NBS Proposed DES[4] is an example of a Shannon Product System, consisting of an algorithm using bitwise permutation, addition mod 2, and substitution in each 64-bit data block in a highly complex sequence, involving multiple iterations through parts of the process, under control of a 64-bit Key (56 data bits plus 8 parity bits). Its basic character is outlined in gross schematic form in Figure 2, which is taken from Branstad.[3]

Shannon represented all simple substitution processes in the form of character set arithmetic (for En-

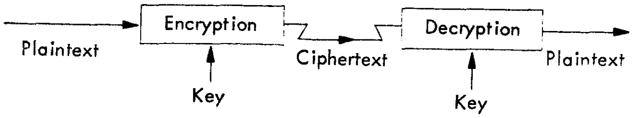
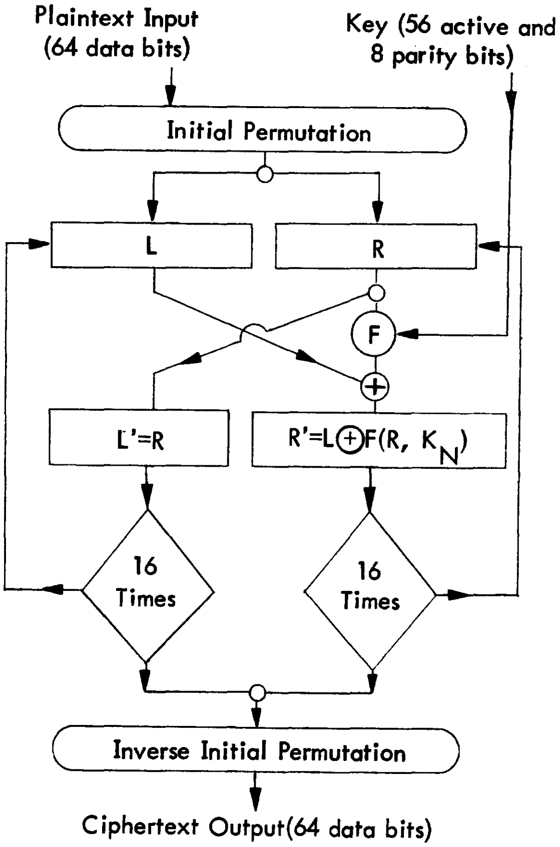


Figure 1—Encryption and decryption



Plaintext Input (64 data bits)
Key (56 active and 8 parity bits)
Initial Permutation
L
R
F
L'=R
R'=L⊕F(R, K_N)
16 Times
16 Times
Inverse Initial Permutation
Ciphertext Output(64 data bits)

Figure 2—Schematic flowchart of DES algorithm

glish alphabetic sets, arithmetic base 26); the Vernam system[5] used bit-by-bit addition modulo 2.

A simple monoalphabetic substitution process, known as a Caesar System (perhaps first introduced to some readers in the form of the Little Orphan Annie Secret Code Wheel), can be represented by

$$y_i = x_i + u$$

where u is a constant and the y_i are the elements of a substitution encoding scheme.

It may be seen by inspection that a Caesar-encoded alphabetic message can be solved by the method of exhaustion, writing it 26 times and choosing one that seems to follow the required language syntax and semantic requirements. Such* processes have not been used by military people since Biblical times.

Vigenère systems[2] use a constantly-changing alphabetic substitution process, stepped synchronously with the message being transformed.

If a single transformation process is stepped to the next sequential substitution using a recirculating transformation representation (initially implemented as a paper tape loop, thus called a "single loop Vigenère system") the transformation can be represented by the sequence

$$a_o, a_1, \ldots, a_\mu$$

A system in which two recirculation transformations are synchronously cascaded can be represented by

$$a_o, a_1, \ldots, a_\mu$$
$$b_o, b_1, \ldots, b_\gamma$$

It is customary to choose μ and γ to be relatively prime.

Tuckerman[6] demonstrated that a general solution to n-loop Vigenère systems could be provided by statistical methods, and showed fully-developed breakings of single- and double-loop Vigenère systems, using real examples.

Shannon proved that a single-loop Vigenère system with infinite "loop" length is unbreakable. Such a scheme is known as "single use code." It requires that the key be at least as long as the message and that the key be kept physically secure.

The significance of the previous paragraph is that the simple Addition Modulo 2 or Exclusive Or algorithm, which is its own inverse, can be used for both encryption and decryption providing that an infinite-length key or an acceptable substitute therefor can be provided and maintained secure. We will use Shannon's designation of such a system as Vernam.

As originally published[5] the Vernam system applied bit-by-bit add-mod-2 (reversible) transformation of a binary message without changing its length.

It may be seen that a similar transformation, also reversible, can be applied through use of sequential logic by merging random noise with the plaintext information to form cipher that will be increased in length by the amount of noise inserted. Subsequent decryption will extract the binary key information and return the cipher to the original plaintext content and length.

Both bitwise operations (Vernam) and sequential logic schemes (merge) depend for their cryptographic strength on the quality of the random number stream used as the Long Key.

The long key information can be retained as physically secure, or can be generated from seed by using a short actual key to start an appropriate random number generator. A third choice is to generate random blocks of key data that can be randomly sequenced to form a pseudo-long-key binary stream. The decision between these implementation approaches should be made in consideration of the required technical performance (in the security sense) and economics of each problem situation.

In the remainder of this paper we will consider only encryption schemes that can be sufficiently strong so that they are economically infeasible to break. We will categorize them into the two broad groups implied in the above discussion:

- Those depending for their strength upon the complexity and effectiveness of a known algorithm, using a nominal-length key, will be called *strong algorithm* systems.
- Those depending for their strength upon the nonpredictability (i.e., the random-bit quality) of a long key, used with relatively trivial algorithms, will be called *long key* systems.

Strong algorithm

Much of the substantive content of the classic Shannon paper[2] and others[7,8] culminating in current algorithm developments has led to what Shannon called Product Systems. Because it has been broadly published and scrutinized, and because current indications are that it satisfies the basic requirements of strength, use of short key, and generality, we suggest that the NBS Proposed DES Algorithm[4] is an appropriate archetype for Strong Algorithm schemes. Its use will be assumed in this section, with the understanding that it could be replaced by a different strong algorithm if such replacement is appropriate.

Proposed Data Encryption Standard (DES)

The National Bureau of Standards (NBS) has selected and published[4] a Proposed Federal Data Encryption Standard. The announcement includes the statement that:

"Data may be protected against unauthorized dis-

* Martin[7] described as a particularly poor example of encryption a method that has been suggested for use with Selectric® typewriter terminals: use scrambled positions for the characters on the type element, and physically secure this removable alphabet.

closure by generating a random key and issuing it to the authorized users of the data. The cipher that has been produced by performing the steps of the encryption algorithm on data using a particular key can only be returned to the original data by use of the decryption algorithm using the identical key. Unauthorized recipients of the cipher who may have the algorithm but who do not have this key cannot derive the original data. A standard algorithm based on a user-generated key thus provides a basis for compatible cryptographic protection of computer data while preventing unauthorized use of the data in cipher form." (The Proposed Standard also states that "Only hardware implementations of the algorithm . . . will be considered as complying with the standard.")

Several manufacturers are developing LSI-chip-based hardware for implementation of the DES algorithm. We are not aware that any of this hardware has been released for sale as of the date of submission of this paper; however, we assume that such release will occur shortly after official confirmation of the Federal Standard.

We have developed exact software Emulators of the DES algorithm for several kinds of hardware, for use as a part of a generalized nonnumeric software element support package. The emulators maintain, and select from, preprocessed key pools. Other parts of the package provide capabilities for data preparation, manipulation, parity bit setting in randomly generated keys, and testing (including optional checking of key parity bits), for use with source programs written in higher procedural languages such as FORTRAN and COBOL.

We anticipate that the special-purpose DES hardware will be much faster in data throughput than the general-purpose versions of the software. The hardware must, of course, be dedicated to particular system functions and consequently cannot be used for general testing and other auxiliary functions without interrupting on-line availability.

NBS has developed two sets of test data for validating implementations of the algorithm:

- What we have called Test A is a set of 24 64-bit key/data pairs designed to demonstrate the power of the algorithm by showing the large effects on cipher of small changes in either datum or key, and the behavior of the encryption process on a variety of bit patterns.
- What we have called Test B is a set of 19 key/data pairs generated as pseudorandom numbers and chosen because the corresponding 19 encryptions reference at least once all of the 512 entries in the "S-box" substitution cipher tables. These 19 pairs were found experimentally by NBS and independently confirmed by us. Correct execution of Test B (in which all cipher produced correspond to presumably correct cipher from a different implementation, and in which all of the S-box table entries are referenced at least once) provides a high order of confidence in the correctness of encryption for any values of key/data.

Result of exercising one of the DES Emulators executing 15 examples from A and B, as described above, is given in Figure 3. In this test output the column headed "CIPHER(ENCRYPTED)" gives the result of machine encryption; the column headed "DECRYPTION OF CIPHER" gives the result of the machine decryption of the machine-produced cipher to recover the original plaintext. All keys and data are shown in hexadecimal notation to correspond with NBS examples.

For all key/data pairs of Tests A and B, the Emulators give results identical to those given by NBS.

From Test A (the first 12 rows of Figure 3), it may be seen that change of a single bit value or a 1-bit shift of a subpattern position within datum or key provides essentially complete change of encrypt/decrypt results. This requires complete accuracy of transmission; conversely, the process can thus be used to display with great sensitivity even slight errors in data entry or transmission.

To show this effect, we repeated Example A9 with the 5th and 6th data digits interchanged (BC becomes CB). Note that the resulting cipher (for Example A25) shows no evident resemblance to the cipher for A9.

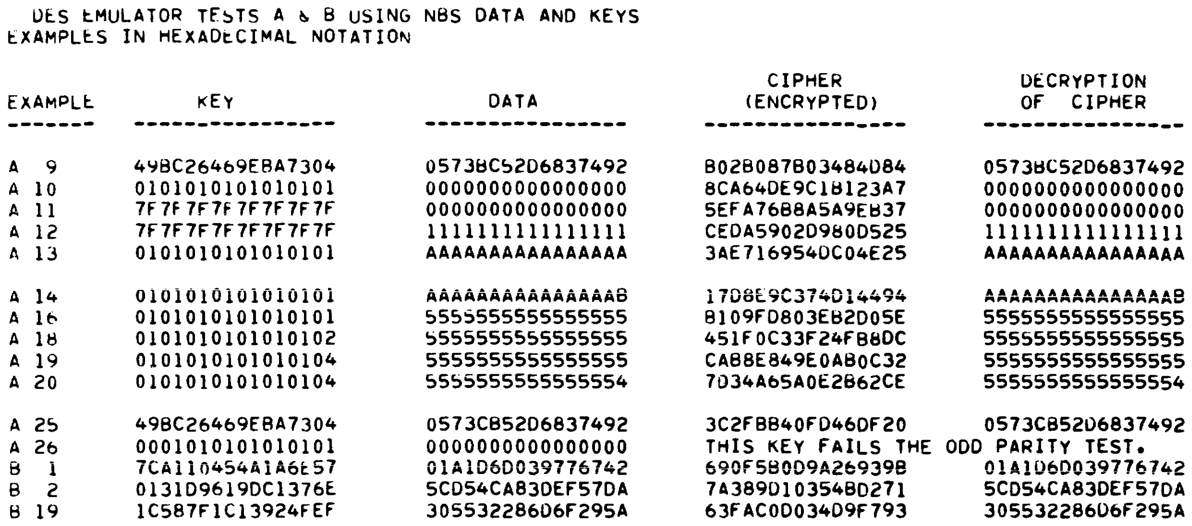
Example A26 is a demonstration of the odd-parity check (optional under the Proposed Standard); it consists of Example A10 modified by giving the first eight key bits even parity.

The general-distribution versions of the DES Emulators use medium-speed, medium-space techniques. NCRYPT/DCRYPT requires less than 9000 bytes on 360/370 or 2200 words on 1108. Execution speed is over 100 encrypts or decrypts per second with machines in the 370/155 or 1108 class.

It may be seen that the strong-algorithm process, if executed entirely by software, will be economical for fairly small data volumes but will be costly for large-volume applications (such as, e.g., encrypting all but control elements of continuous high-speed data streams or sizable data bases). For large-scale applications, the long-key methods outlined in the next section (after either a time delay for about a million machine instructions executed in initial (from seed) startup of the high-performance random number generator provided, or time to load a 1042-word restart table) operate orders of magnitude faster than the Emulator.

Applications of the software DES emulator include:

- Testing of application ideas and methods before hardware is available;
- Debugging and production testing of programs independently of on-line hardware;



```
  DES EMULATOR TESTS A & B USING NBS DATA AND KEYS
EXAMPLES IN HEXADECIMAL NOTATION

                                                            CIPHER                  DECRYPTION
EXAMPLE        KEY                      DATA              (ENCRYPTED)               OF  CIPHER
-------   ----------------        ----------------     ----------------          ----------------

A   9     49BC26469EBA7304        0573BC52D6837492     B02B087B03484D84          0573BC52D6837492
A  10     0101010101010101        0000000000000000     8CA64DE9C1B123A7          0000000000000000
A  11     7F7F7F7F7F7F7F7F        0000000000000000     5EFA76B8A5A9EB37          0000000000000000
A  12     7F7F7F7F7F7F7F7F        1111111111111111     CEDA5902D980D525          1111111111111111
A  13     0101010101010101        AAAAAAAAAAAAAAAA     3AE716954DC04E25          AAAAAAAAAAAAAAAA

A  14     0101010101010101        AAAAAAAAAAAAAAAB     17D8E9C374D14494          AAAAAAAAAAAAAAAB
A  16     0101010101010101        5555555555555555     B109FD803EB2D05E          5555555555555555
A  18     0101010101010102        5555555555555555     451F0C33F24FB8DC          5555555555555555
A  19     0101010101010104        5555555555555555     CA88E849E0AB0C32          5555555555555555
A  20     0101010101010104        5555555555555554     7D34A65A0E2B62CE          5555555555555554

A  25     49BC26469EBA7304        0573CB52D6837492     3C2FBB40FD46DF20          0573CB52D6837492
A  26     0001010101010101        0000000000000000     THIS KEY FAILS THE ODD PARITY TEST.
B   1     7CA110454A1A6E57        01A1D6D039776742     690F5B0D9A26939B          01A1D6D039776742
B   2     0131D9619DC1376E        5CD54CA83DEF57DA     7A389D10354BD271          5CD54CA83DEF57DA
B  19     1C587F1C13924FEF        30553228D6F295A      63FAC0D034D9F793          305532286D6F295A
```

Figure 3—DES emulator encryption/decryption results: Tests A and B

- Preparation and evaluation, by manufacturers, of hardware design test data;
- Testing (both validation and maintenance) of installed hardware;
- Operational encryption/decryption where the Federal Standard is not applicable; and
- Bidirectional authentication procedures (See, e.g., Reference 1).

Long-key systems

Any long-key system depends upon the use of a key stream that must be assumed to be physically secure. The key stream can be provided in two basic ways:

(a) It can be generated from "seed" (i.e., a key unique to that key stream) in a fully-deterministic process when used for either encryption or decryption or both; or

(b) It can be generated, stored, copied as may be required, and played back from the stored form when and where needed.

In case (a), decryption of a particular ciphertext data stream requires that a key-stream generator (software or hardware) identical to that used in encryption of that stream must be available at the time and place of key stream entry for decryption.

Physical security must be provided for the key to be used: in case (b), this will require security of tape or diskpack; in case (a), only the seed need be kept secure. Obviously, physical and geographical considerations will affect the generate-or-playback choice.

An important consideration is that long messages would be in hazard from even trivial communication or other hardware errors; loss of absolute synchronization would generate chaos, effectively preventing recovery of plaintext beyond the point at which, say, a one-bit loss occurred.

The very vulnerability of ciphertext, as noted here, provides a potentially useful and highly sensitive detection scheme: even small errors in ciphertext transmission will result in gross and obvious format and other changes in decrypted output.

Use of message blocking (we have chosen 64 bits as the standard block length corresponding to DES practice[3,4]), and careful block numbering and accounting, prevents loss of more than a single block for a single small error and provides an audit trail for recovery.

As noted in References 3 and 4, validity of the encryption process as secure in and of itself, depending only upon key security, requires that the system not depend upon secrecy of an algorithm or of hardware/software configuration.

With either of the long-key methods discussed below, in order to maximize security and integrity of the encryption process, "leakage" of key stream control information elements should be inhibited in spite of the simplicity of the actual encryption algorithms. Thus, it is desirable that the appearance of cyclic or unchanging (i.e., transparent) bit patterns in plaintext (see, e.g., the all-zeroes and alternate-ones data of Examples A10 and A13 of Figure 6) be suppressed by compression or other means.

Long-key generation from seed

One effective means of key stream generation is a process that produces uniformly distributed random numbers. The process used here is a computer program quasirandom number stream generator of the

Tausworthe-Lewis-Payne bitwise linear recurrence modulo 2 type, of which the developmental background and characteristics are outlined in the Appendix.

Chaitin[9] expressed proof that a truly random string cannot be specified by an information string shorter than itself. It may be inferred that a perfect single-use code cannot have its key stream generated from a short seed expanded by any program of limited complexity.

We believe, however, that despite its finite complexity a publicly known quasirandom number stream generator of long period and good statistical performance, such as that described in the Appendix, when operating on a secret seed, can produce a sufficiently close approximation to a random key stream so that the fundamental objective of the present paper is met: the resulting encryption will be economically infeasible to break.

Figure 4 shows schematically the process by which the quasirandom number generator produces a 64-bit-block key stream starting from a 64-bit (or less) seed, continually accessing a recirculating seed matrix of about a thousand words as outlined in the Appendix.

Initial startup of the generator from a seed requires a half million machine instructions to be executed, after which the process proceeds at a speed corresponding to only a dozen machine instructions executed per number generated.

Restart of the process can be accomplished readily through reloading of the recirculating seed matrix. Backwards re-generation from a previous checkpoint, as is required for some system problems, offers no difficulties; the generator can operate in either direction. Pre-iteration count is a user-controlled parameter.

For encryption purposes this Tausworthe-Lewis-Payne generator offers the advantage that a large amount of information (521 64-bit blocks) would be needed to initiate or restart a quasirandom string. There is no explicit way of identifying the initially-used element of the generated string, other than knowing the actual seed and the starting iteration count. The fact that both seed and period are of great length permits both Vernam and merge/extract encipherments to have considerable cryptographic strength.

On a 360/65, after restart (time to load the current seed string matrix) or startup (2.3 seconds for a pre-iteration count of 20,000p), the generation of 64-bit unsigned quasirandom integers takes 24.1 milliseconds per thousand numbers.

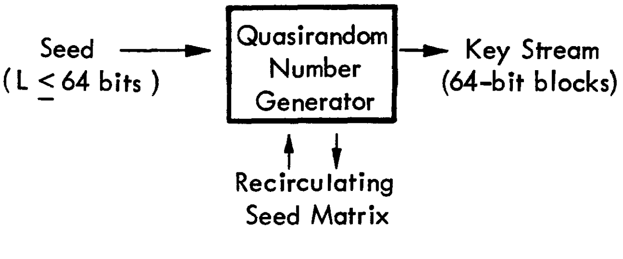


Figure 4—Long-key generation from seed

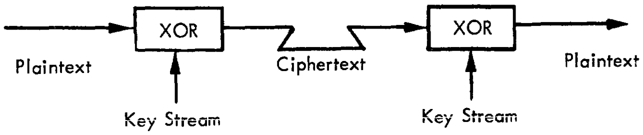


Figure 5—Long-key Vernam (bitwise) logic

The period of this Tausworthe-Lewis-Payne type generator is essentially infinite (viz., $2^{521}-1$). Its gross performance has been checked in dimensionality up to 8-distributivity. Small changes in seed cause large changes in generated key stream.

Long-key Vernam (bitwise)

The classical Vernam single-use-code concept, implemented explicitly, is shown schematically in Figure 5.

As noted previously, this scheme[11] does not change message length.

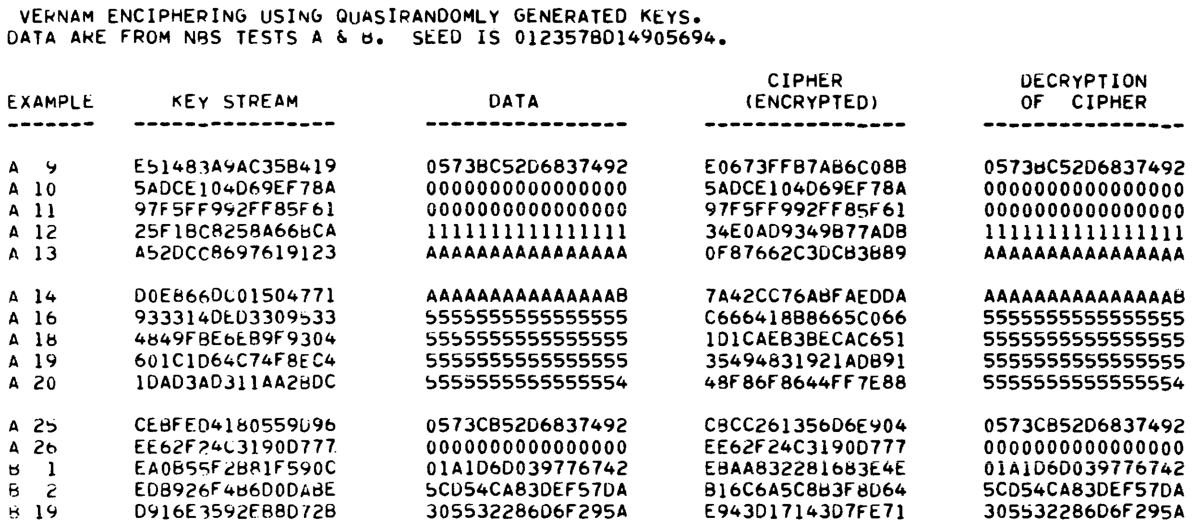
The building-block approach uses prefabricated software elements for executing the encryption and control logic and for data preparation and testing.

A Vernam test is shown in Figure 6. This test uses as plaintext input data, read from cards, the 15 examples used in Figure 3 above to test the DES Emulator, consisting of 13 examples of NBS "Tests A and B" plus slight modifications of two of the Test A examples. The key stream was generated by the TLP quasirandom number generator from the 64-bit seed 012357BD14905694. Speed of this encryption or decryption on a 360/65 is one 64-bit block per 12.3 microseconds.

Long-key sequential logic

The capability for controlled merging and extracting of noise into and from binary information streams has several potential uses and should be considered in a comprehensive encryption plan:

- Provide a high level of cryptographic protection when used with a long quasirandom number control stream and appropriately generated noise data;
- Provide an intermediate level of cryptographic protection when used with shorter or recycling control stream and a noise data stream of any length;
- Suppress redundancy or cyclic patterns in plaintext or in ciphertext, for the purpose of raising the cryptographic strength of DES or Vernam-type high-level cryptography systems; and
- Permit, in combination with the DES and/or Vernam techniques, the implementation of multiple-level cryptography systems of controllable



```
 VERNAM ENCIPHERING USING QUASIRANDOMLY GENERATED KEYS.
DATA ARE FROM NBS TESTS A & B.  SEED IS 01235780149O5694.

                                                        CIPHER             DECRYPTION
EXAMPLE      KEY STREAM            DATA               (ENCRYPTED)          OF  CIPHER
-------   ----------------   ----------------   ----------------   ----------------

A  9      E51483A9AC358419   0573BC52D6837492   E0673FFB7AB6C08B   0573BC52D6837492
A 10      5ADCE104D69EF78A   0000000000000000   5ADCE104D69EF78A   0000000000000000
A 11      97F5FF992FF85F61   0000000000000000   97F5FF992FF85F61   0000000000000000
A 12      25F1BC8258A66BCA   1111111111111111   34E0AD9349B77ADB   1111111111111111
A 13      A52DCC8697619123   AAAAAAAAAAAAAAAA   0F87662C3DCB3B89   AAAAAAAAAAAAAAAA

A 14      D0E866DC01504771   AAAAAAAAAAAAAAAB   7A42CC76ABFAEDDA   AAAAAAAAAAAAAAAB
A 16      933314DED3309533   5555555555555555   C666418B8665C066   5555555555555555
A 18      4849FBE6EB9F9304   5555555555555555   1D1CAEB3BECAC651   5555555555555555
A 19      601C1D64C74F8EC4   5555555555555555   35494831921ADB91   5555555555555555
A 20      1DAD3AD311AA2BDC   5555555555555554   48F86F8644FF7E88   5555555555555554

A 25      CEBFED4180559D96   0573CB52D6837492   C8CC261356D6E904   0573CB52D6837492
A 26      EE62F24C3190D777   0000000000000000   EE62F24C3190D777   0000000000000000
B  1      EA0B55F2B81F590C   01A1D6D039776742   EBAA8322816B3E4E   01A1D6D039776742
B  2      ED8926F4B6D0DABE   5CD54CA83DEF57DA   816C6A5C8B3F8D64   5CD54CA83DEF57DA
B 19      D916E3592E88D72B   30553228606F295A   E943D17143D7FE71   30553228606F295A
```

Figure 6—Vernam encryption/decryption results

complexity and with controllable strength/cost tradeoff at each level.

The encryption and decryption processes can be controlled from fixed mask (MASK=constant); lists or tables of masks (MASK=array variable); or from a recycling or open-ended mask stream, with corresponding levels of complexity and security.

Use of a constant merge control mask produces weak cipher that has recognizable patterns, especially when used with simple bit pattern plaintext and examined in binary form; a variable mask produces higher-grade cipher.

A sequential logic test is shown in Figure 8. In this case a constant quasirandom 64-bit key (actually one of the generated keys, chosen because it happens to contain exactly 32 one-bits and 32 zero-bits, generated from the same seed used in the Vernam test above) is used as the control mask for noise (the same 15 generated quasirandom keys) to be merged with plaintext to form a kind of cipher. This mask, because of the 50/50 one/zero mask ratio, produces cipher just twice the length of the plaintext. The two sequential 64-bit cipher blocks produced from each plaintext block are shown together in the double column headed CIPHER (ENCRYPTED). The decrypted plaintext, recovered by extraction of the noise stream under control of the same mask, is shown in the right-hand column labeled DECRYPTION OF CIPHER.

The requirements for bitwise testing and processing make this sequential process slower than the Vernam process.

PROGRAM PROTECTION

The protection of proprietary programs from unauthorized use, copying, or alteration is a largely unexploited area of application of cryptography that offers fascinating opportunities for strategy and counter-strategy development.

Protection technology for file and system access, as outlined previously, provides the basic mechanisms for program transformation, detection of alteration, and audit trail development.

A basic decision that must be made at the outset of development of a program protection method is whether personnel and organizations having some level of authorization for system software maintenance are to be considered part of the world against which protection (from errors and omissions as well as malfeasance) is sought. If the answer is yes, the problem set escalates in difficulty and reasonable bounds on the protection objectives must be established.

APPENDIX

Background for a 64-bit quasirandom number string generator

The "mid-squares" random number generators used by pioneers Von Neumann and others in the late 1940s rapidly degenerated (to zeroes or cycles of short period) when used for long string generation. Like physical (e.g., electronic noise) generators, they were

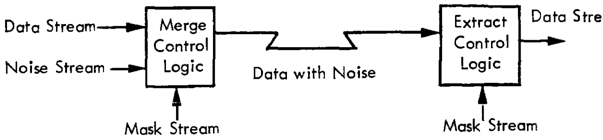
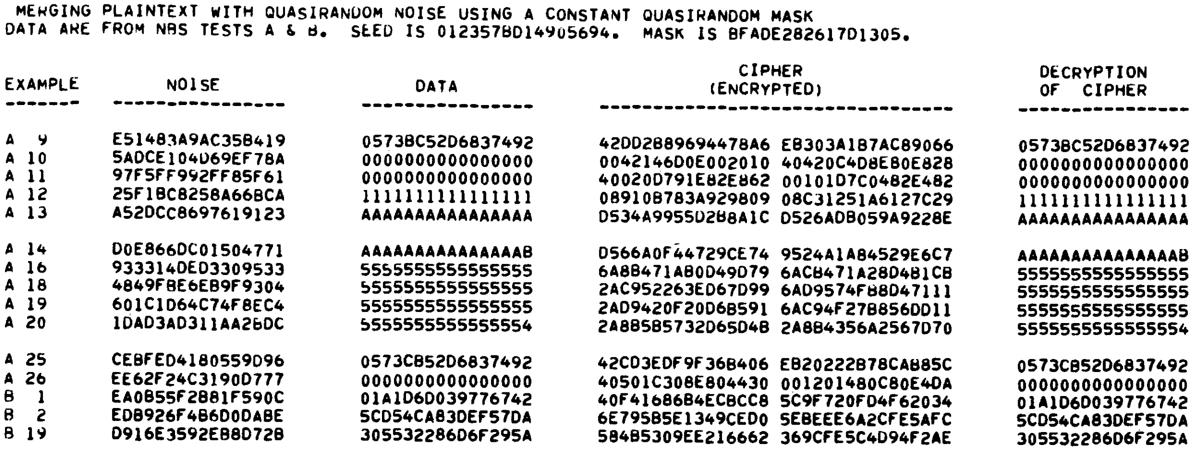


Figure 7—Long-key sequential logic



```
 MERGING PLAINTEXT WITH QUASIRANDOM NOISE USING A CONSTANT QUASIRANDOM MASK
DATA ARE FROM NBS TESTS A & B.  SEED IS 012357BD14905694.  MASK IS BFADE282617D1305.

                                                                  CIPHER                   DECRYPTION
EXAMPLE        NOISE                DATA                       (ENCRYPTED)                 OF  CIPHER
-------   ----------------    ----------------    ---------------------------------    ----------------

A   9     E51483A9AC35B419    0573BC52D6837492    42DD2B89694478A6 EB303A1B7AC89066    0573BC52D6837492
A  10     5ADCE104D69EF78A    0000000000000000    0042146D0E002010 40420C4D8E80E828    0000000000000000
A  11     97F5FF992FF85F61    0000000000000000    40020D791E82E862 00101D7C0482E482    0000000000000000
A  12     25F1BC8258A66BCA    1111111111111111    08910B783A929809 08C31251A6127C29    1111111111111111
A  13     A52DCC8697619123    AAAAAAAAAAAAAAAA    D534A9955D2B8A1C D526ADB059A9228E    AAAAAAAAAAAAAAAA

A  14     D0E866DC01504771    AAAAAAAAAAAAAAAB    D566A0F44729CE74 9524A1A84529E6C7    AAAAAAAAAAAAAAAB
A  16     933314DED3309533    5555555555555555    6A8B471AB0D49D79 6ACB471A28D4B1CB    5555555555555555
A  18     4849FBE6EB9F9304    5555555555555555    2AC952263ED67D99 6AD9574FB8D47111    5555555555555555
A  19     601C1D64C74F8EC4    5555555555555555    2AD9420F20D6B591 6AC94F27B856DD11    5555555555555555
A  20     1DAD3AD311AA26DC    5555555555555554    2A885B5732D65D4B 2A884356A2567D70    5555555555555554

A  25     CEBFED4180559D96    0573CB52D6837492    42CD3EDF9F36B406 EB20222B78CA885C    0573CB52D6837492
A  26     EE62F24C3190D777    0000000000000000    40501C308E804430 001201480C80E4DA    0000000000000000
B   1     EA0B55F2881F590C    01A1D6D039776742    40F41686B4ECBCC8 5C9F720FD4F62034    01A1D6D039776742
B   2     EDB926F4B6D0DABE    5CD54CA83DEF57DA    6E795B5E1349CED0 5EBEEE6A2CFE5AFC    5CD54CA83DEF57DA
B  19     D916E3592EB8D72B    305532286D6F295A    584B5309EE216662 369CFE5C4D94F2AE    305532286D6F295A
```

Figure 8—Sequential encryption/decryption: Tests A and B

unsuitable for many modern computing purposes. They were largely displaced by generators (programs) of many designs using congruential methods.

Generators of one type, suggested by Lehmer[30] in 1951 and known as "linear congruential", can be described by the recursive formula

$$x_{i+1} \equiv ax_i + c \pmod{m}$$

where $\{x_i\}$ is the sequence generated. The parameters are the seed (x_0), the multiplier (a), the additive constant (c) and the modulus (m). This is called the multiplicative congruential method when $c=0$, and the mixed congruential method when $c \neq 0$. The x_i's are integers from 0 to $m-1$. A large value of m is needed to avoid trivial sequences with small period. For example, if $m=2$, the sequence is either a constant or an alternating 0, 1, 0, 1, ... Uniform numbers between 0 to 1 are obtained by $u_i = \frac{x_i}{m}$.

Generators of a second type, known as "additive congruential," can be described by

$$x_i \equiv x_{i-q} + x_{i-p} \pmod{m}.$$

Both are special cases of the general linear congruential method:

$$x_i \equiv a_1x_{i-1} + a_2x_{i-2} + \ldots . + a_px_{i-p} + c \pmod{m}.$$

The additive method has properties sufficiently different from the linear as to make it worth studying separately. A large modulus is not needed to ensure a long period; arbitrarily long period sequences can be generated with $m=2$, provided judicious choices are made for p and q.

Both linear and additive methods are described in Knuth Volume 2.[10] Knuth observes that a good sequence of random numbers can be obtained by the additive congruential method, or more generally, by

$$x_i \equiv a_1x_{i-1} + \ldots + a_kx_{i-k} \pmod{m}$$

provided that m is prime and the polynomial $x^k - a_1x^{k-1} - \ldots - a_k$ is primitive modulo m. Such generators with $m=2$ were proposed by Tausworthe[29] in 1965 and are referred to as linear recurrence modulo 2. In this case, the a_i's are 0 or 1; we may write

$$x_i \equiv x_{i-q_1} + \ldots + x_{i-q_r} \pmod{2}$$

and the polynomial becomes $x^{q_r} + \ldots + x^{q_1} + 1$ (note $-1 \equiv 1 \pmod{2}$)). In the simplest case, $r=2$ and the polynomial is a trinomial of the form $x^p + x^q + 1$. Such a generator is simple to implement on many modern electronic computers: addition modulo 2 is provided as the machine operation Exclusive Or.

An additive congruential generator not of this type was implemented by Carroll and McLelland[11] using suggestions made by Green, Smith, and Klem.[12]

Green, Smith and Klem showed that such a generator did not produce a very random sequence; Knuth rejected the Tausworthe generator as a poor source of random strings of bits; however, strings of consecutive bits from a Tausworthe sequence do not yield poor results when larger and more carefully chosen values of p and q are used than those tested by Knuth, and the string sequence can be further improved by using a technique recently proposed by Lewis and Payne.[13]

Statistical tests and criteria

Many tests for a would-be random number generator have been proposed and used. Several tests are described by Knuth:

(1) Equidistribution or Frequency Test: One counts the number of times a member of the sequence falls into a given interval. The number should be approximately the same for intervals of the same length if the sequence is uniform.

(2) Serial Test: This is a higher-dimensional version of the equidistribution test. One counts the num-

ber of times a k-tuple $(x_{ki}, x_{ki+1}, \ldots, x_{ki+k-1})$ of members of the sequence falls into a given k-dimensional cell. If this test is passed, the sequence is said to be k-distributed. It is not practical to divide the coordinate axes into too many subintervals when k is large, say, $k>3$. Other tests of k-distributivity are:

(2a) Poker Test: One considers groups of k members of the sequence and counts the number of distinct values represented in each group.

(2b) Maximum [Minimum] of k: One plots the distribution of the function max [min] $(x_{ki}, \ldots, x_{ki+k-1})$.

(2c) Sum of k: This is similar to tests described by Knuth. The same tests can be made for any function of k numbers, provided one can calculate an expected distribution for the function. This test is explicitly mentioned by MacLaren and Marsaglia[14] as well as by Lewis and Payne, who refer to it as the Yules Test. Several authors use $\sum_{1}^{k} x_i^2$, which is commonly called "the d^2 test".

(3) Gap Test: One plots the distribution of gaps in the sequence of various lengths, i.e., consecutive members $x_i, x_{i+1}, \ldots, x_{i+r}$, such that all members fall into a given interval, while the immediately preceding and succeeding members do not. A special case is where the interval is the set of numbers above (or below) the mean, in which case it is called the Runs Above (Below) the Mean Test.

(4) Runs Test: One plots the distribution of maximal ascending (descending) runs of various lengths. This test (now called the Runs Up/Runs Down Tests) was mentioned by Moshman[15] in the first random number generator paper ever published in an ACM serial.

(5) Coupon Collector's Test: One chooses a suitably small integer d and divides the universe into d intervals. Then each member of the sequence falls into one such interval. One plots the distribution of runs of various lengths required to have all d intervals represented.

(6) Permutation Test: One studies the order relations between the members of the sequence in groups of k. Each of the k! possible orders should occur about equally often. If the universe is large, the probability of equality is small; otherwise, equal members may be disregarded.

(7) Serial Correlation Test: One computes the correlation coefficient between consecutive members of the sequence. This gives the serial correlation for lag 1. Similarly, one may get the serial correlation for lag k by computing the correlation coefficient between x_i and x_{i+k}. This is to show that the members of the sequence are independent.

Other tests have been proposed and used. Lewis and Payne[13] ran a Conditional Bit Test, which tested the independence of each bit in a string from the others, as well as a Fourier Transform Test, also used by Coveyou and McPherson[16] in 1967 and by Lewis, Goodman and Miller.[17] The latter was facilitated by the Fast Fourier Transform algorithm introduced by Cooley and Tukey.[18]

All of the pre-Tausworthe papers referenced here proposed some sort of linear congruential generator. Those who subjected their generators to exhaustive tests admitted that some of the tests failed. Some only ran the simplest tests (e.g., equidistribution and serial test for pairs), and passed; however, in typical cases the same generators were later shown to fail some other test. The generators differed mainly in the choice of such parameters as multiplier, additive constant, and modulus. The modulus was usually the largest number that could be represented on some machine (most often 2^{35}), or a prime less than that number. The present problem required a 64-bit random number generator, which none of the above papers considered. Many authors seemed to choose various parameters arbitrarily and pick the ones that passed the most tests, although in some cases broad guidelines were given (but, in general, were not shown to be useful). One formula giving an approximation to the serial correlation of a linear congruential sequence was published by Coveyou.[19] In certain cases his approximation was a poor one, but Greenberger[20] corrected this flaw by adding another term. Some theoretical and empirical work by Marsaglia [21] in 1968 showed that all linear congruential generators suffered from poor higher-dimensional distributions. There was then little hope of producing one generator that passed all tests.

A paper by Martin-Löf[22] showed, using methods of recursive function theory, that there was a universal test for uniform random number sequences and that almost all sequences (in the sense of measure theory) passed it. Although this paper is quite abstract and not of much practical use in constructing a good random number generator, it did provide some hope that it could be done.

Knuth mentioned a property that seems to come close to the concept of a universal test, namely, complete equidistributivity, or ∞-distributivity: If a sequence is k-distributed, it is r-distributed for $r<k$. A sequence ∞-distributed if it is k-distributed for all k.

A sequence that is ∞-distributed passes all the other tests we have considered. Such a sequence was constructed by Knuth,[23] but, as he observed, his sequence is not of much use in machine generation of random numbers because it takes too long to converge to the desired properties. It was formed by starting with a short 1-distributed sequence, followed by a slightly longer 2-distributed sequence, etc.

The papers after Tausworthe's own paper that describe Tausworthe sequences show increasingly good results: it seems that such sequences can be constructed to pass more of the tests than the linear congruential generators. Whittlesey[24] showed that linear congruential generators that had passed other tests failed some autocorrelation tests he performed with lags from 1 to 50. These tests are related to the serial correlation test.

The Tausworthe sequence passed Whittlesey's test; in fact, it can be shown to pass these and many other tests on purely theoretical grounds. We found such analytical support to be lacking in the papers proposing linear congruential generators. Tootill, Robinson and Adams[25] showed that a Tausworthe sequence had good Runs Up and Down properties, and Tootill, Robinson and Eagle[26] showed that a Tausworthe sequence they generated was indistinguishable, by empirical tests, from one that was ∞-distributed.

It appears to be consensual among recent authors that, in generating a Tausworthe sequence, it is best to use a trinomial x^p+x^q+1 whose degree p is such that 2^p-1 is relatively prime to various parameters of the tests. Also, better results are obtained if decimation is used, which means that one does not pick consecutive groups of bits from the sequence but rather spaces them out; and the amount of spacing should be relatively prime to 2^p-1.

One would expect the best results from a Tausworthe generator if the degree p were such that 2^p-1 were itself a prime number. Such primes are called Mersenne primes, and p is called a Mersenne exponent. A table of primitive trinomials whose degree is a Mersenne exponent was published by Zierler.[33] It included the 23 then known Mersenne exponents, which form a consecutive set, the largest of which is 11213. The largest one having a primitive trinomial is 9689.

The most promising generator we have found seems to be the one described by Lewis and Payne.[13] They use a technique that appears to be superior to decimation: to generate r-bit numbers, each of the r bits is chosen from a different part of the same Tausworthe sequence with a constant gap between bits. Lewis and Payne suggest that a gap of at least 100p should be sufficient. They also suggest that the sequence will be k-distributed for $k \leq rp$.

Our Tausworthe-Lewis-Payne (TLP) generator uses the trinomial $x^{521}+x^{32}+1$ to generate 64-bit numbers that are 8-distributed and have good k-distributivity for $k>8$. The period of the sequence is $2^{521}-1$.

The degree-521 TLP generator requires 521 bits (not all zero) to start, which must somehow be expanded from the initial 64-bit seed. If these 521 bits are "not very random", the next few members of the sequence will also have this weakness; however, Lewis and Payne suggest that about 5000p (here, 2.6 million) bit-iterations of a Tausworthe generator should suppress such non-randomness. Our initial statistical tests appear to confirm that suggestion if iteration count is increased to 20,000p. We have considered but not implemented use of the primitive trinomial $x^{89}+x^{38}+1$ with the TLP method for generating the seed string. This two-level TLP may reduce the number of iterations required.

We have used the prefix "quasi-" which, according to Webster's 3ID, means "seemingly, almost" together with the word "random" to identify effective generators. Most authors have used the prefix "pseudo-" which, per Webster, connotes "false, sham, feigned, fake, counterfeit, spurious". Subsequent test results have shown, alas, that "pseudo" was often an appropriate descriptor. It is hoped that the performance of the generator described here will prove, on further testing, to have justified our use of the term "quasi-random number generator".

We gratefully acknowledge discussions on applicability of congruential methods with Juncosa,[28] whose work was used by many subsequent authors, and on primitive trinomials with Tausworthe,[29] whose basic concept underlies the most promising current developments.

REFERENCES

1. Fiestel, H., W. A. Notz and J. L. Smith, "Some Cryptographic Techniques for Machine-to-Machine Data Communications," *Proceedings IEEE* 63, November 11, 1975, pp. 1545-1554.
2. Shannon, C. E., "Communication Theory of Secrecy Systems," *Bell Sys. Tech. Journal*, October 1949, pp. 656-715.
3. Branstad, D. K., *Draft Guidelines for Implementing and Using the NBS Data Encryption Standard*, National Bureau of Standards, November 10, 1975.
4. *National Bureau of Standards Announcement of Proposed Federal Data Encryption Standard, Federal Register*, March 1975 (40FR12607).
5. Vernam, G. S., "Cipher Printing Telegraph Systems for Secret Wire and Radio Telegraphic Communications," *Journal AIEE*, 1926, pp. 469-481.
6. Tuckerman, B., *A Study of the Vigenère-Vernam Single- and Multiple-Loop Enciphering Systems*, IBM Research Report RC 2879 (#13538), May 14, 1970 (Mathematics).
7. Martin, J., *Security, Accuracy and Privacy in Computer Systems*, Prentice-Hall 1973.
8. Feistel, H., *Cryptographic Coding for Data-Bank Privacy*. IBM Research Report RC 2827, (#13260), March 18, 1970.
9. Chaitin, G. J., "Information-Theoretic Limitations of Formal Systems," *JACM* 21/3, July 1974, pp. 403-424.
10. Knuth, D., *The Art of Computer Programming*, Vol. 2, "Seminumerical Algorithms," Addison-Wesley, 1969.
11. Carroll, J. M. and P. M. McLelland, "Fast Infinite-Key Privacy Transformation for Resource-Sharing Systems," *Proceedings FJCC*, '70, AFIPS, Vol. 37, pp. 223-230.
12. Green, B. F., J. E. K. Smith and L. Klem, "Empirical Tests of an Additive Random Number Generator," *JACM*, 6, 1959, pp. 527-537.
13. Lewis, P. A. W. and W. H. Payne, "Generalized Feedback Shift Register Pseudorandom Number Algorithm," *JACM* 20, 1973, pp. 456-468.
14. MacLaren, M. D. and G. Marsaglia, "Uniform Random Number Generators," *JACM* 12, 1965, pp. 83-89.
15. Moshman, J., "The Generation of Pseudo-Random Numbers on a Decimal Calculator," *JACM* 1, 1954, pp. 88-91.
16. Coveyou, R. R. and R. D. MacPherson, "Fourier Analysis of Uniform Random Number Generators," *JACM* 14, 1967, pp. 100-119.
17. Lewis, P. A. W., A. S. Goodman and J. M. Miller, "A Pseudo-Random Number Generator for the System/360," *IBM Syst. J.* 8, 1973, pp. 456-468.

18. Cooley, J. W. and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. of Comp.* 19, 1965, pp. 297-301.

19. Coveyou, R. R., "Serial Correlation in the Generation of Pseudo-Random Numbers," *JACM* 7, 1960, pp. 72-74.

20. Greenberger, M., "An a priori Determination of Serial Correlation in Computer Generated Random Numbers," *Math. of Comp.* 15, 1961, pp. 383-389.

21. Marsaglia, G., "Random Numbers Fall Mainly in the Planes," *Proc. Nat. Acad. Sci.* 6, 1968, pp. 25-28.

22. Martin-Löf, P., "The Definition of Random Sequences," *Information and Control* 9, 1966, pp. 602-619.

23. Knuth, D., "Construction of a Random Sequence," *BIT* 5, 1965, pp. 246-250.

24. Whittlesey, J. R. B., "A Comparison of the Correlational Behavior of Random Number Generators for the IBM 360," *CACM* 11, 1968, pp. 641-644.

25. Tootill, J. P. R., W. D. Robinson, and A. G. Adams, "The Runs Up-and-Down Performances of Tausworthe Pseudo-Random Number Generators," *JACM* 18, 1971, pp. 381-399.

26. Tootill, J. P. R., W. D. Robinson and J. Eagle, "An Asymptotically Random Tausworthe Sequence," *JACM* 20, 1973, pp. 469-481.

27. Zierler, N. and J. Brillhart, "On Primitive Trinomials, (Mod2)," II, *Information and Control* 14, 1969, pp. 566-569.

28. Juncosa, M. L., *Random Number Generation on the BRL High Speed Computing Machine*, BRL Report No. 855, 1953.

29. Tausworthe, R. C., "Random Numbers Generated by Linear Recurrence Modul Two," *Math. of Comp.* 19, 1965, pp. 201-209.

30. Lehmer, D. H., *Mathematical Methods in Large-Scale Computing Units*, Ann. Comp. Lab. Harvard U. 26, 1951, pp. 141-146.

31. U.S. Patents (assigned to IBM Corporation and underlying the DES product block cipher algorithm): Feistel, *Block Cipher Cryptographic System*, No. 3, 798,359 of 1974 and Smith, *Recirculating Block Cipher Cryptographic System*, No. 3, 796,830 of 1974.

32. Skatrud, R. O., "A Consideration of the Application of Cryptographic Techniques to Data Processing," *Proc. FJCC* '69, AFIPS, Vol. 35, pp. 111-117.

33. Zierler, N., Primitive Trinomials Whose Degree is a Mersenne Exponent," *Information and Control* 15, 1969, pp. 67-69.

34. Zierler, N. and J. Brillhart, "On Primitive Trinomials (Mod 2)," *Information and Control* 13, 1968, pp. 541-554.

Analysis of secret functions with application to computer cryptography

by INGEMAR INGEMARSSON
Linköping University
Linköping, Sweden

ABSTRACT

In computer cryptography we cannot avoid that data and the corresponding encrypted data can be read by an outside observer. The information contained in these observations may be used to decrypt parts of encrypted data or ultimately to identify the key in the cryptographic transformation. In this paper we have analyzed this situation using the concepts of information theory. The result shows that in most cases it is theoretically possible for an outside observer to identify the key after very few observations. As this must be avoided we have to rely on computational complexity in the process of deriving the key. This is achieved by using one-way functions which are practically impossible to invert.

INTRODUCTION

Computer cryptography differs from communication cryptography in two respects: (i) A particular set of data is used more than once and by several users. (ii) Data are processed by the computer. These two differences impose restrictions on the type of cryptologic transformations suitable for use in an electronic data processing (EDP) system. In this paper we will focus our attention on the problems caused by property (i) above.

Communication cryptography was analyzed by Shannon 1949.[1] His model includes an information source (a stochastic process) with known statistics. The information from the source is encrypted and then observed by an outside observer. The goal of the observer is to derive the original information and/or identify the encryption transformation. Our approach is somewhat different. When several users have access to the same data, the encryption transformation for the particular set of data is preferably fixed, at least for some time. In an EDP system it is also realistic to assume that a set of data and the corresponding cryptogram is known to an outside observer. The observer may use this knowledge to facilitate decryption of other cryptograms, encrypted with the same transformation. He may also ultimately be able to identify the encryption transformation. We want to investigate his chances to work in these directions. A related problem, from the legitimate user's point of view, is: How do we avoid that an outside observer gets the ability to decrypt stored cryptograms? Our approach originates in information theory. The encryption or decryption transformation (or algorithm) may be regarded as a set of known functions of the input variable. The function is chosen by the key which is unknown. From the beginning an outside observer does not know which function is actually realized. To him each function has the same probability.

When he observes the input and the corresponding output he receives some information about the actual function. The number of possible functions is decreased. To what extent does this help the observer if he wants to estimate the output corresponding to another input? We may formulate the question more precisely this way: Before the observation the uncertainty, measured as entropy, is H_0 bits. If no information as how to estimate other outputs is conveyed by the first observation, then the uncertainty remains unchanged. The entropy connected with the second observation is still H_0. It may sound as a good design objective to keep the uncertainty (the entropy) unchanged after several observations. Unfortunately this is limited by one of the main results of this paper.

$$\sum_{i=0}^{\infty} H_i = \log M \qquad (1)$$

Here H_i is the entropy after i observations and M is the number of encryption or decryption transformations, i.e., the number of keys. As is seen from equation (1) the requirement that the entropy should remain constant must be limited to a finite (and perhaps low!) number of observations. After that the entropy is zero, i.e., the observer knows exactly which function is actually chosen, i.e., he knows the key!

If we want to avoid this fallacy the sequence of entropies

$$\{H_i\}_{i=0}^{\infty}$$

must decrease. This means that the difficulty of the observer to estimate outputs for given inputs is decreasing with the number of observations! Obviously we have to compromise to obtain reasonable protection of the key and reasonable low chance to estimate the outputs.

However, when typical figures are put in the equations the result is most unsatisfying for the designer. If the model is accurate, it is too easy to break the system. The way out of this dilemma is of course to build a system for which this model cannot be used. The main point here is that we have not taken into account the computational problem involved in estimating the output for a given input starting from the knowledge from several observations. Hence the theory shows that we have to rely on computational complexity when designing computer cryptographic systems. The complexity requirement is preferably formulated in terms of *one-way functions.* Such a function is easy to compute but its inverse is not computable in a reasonable amount of time.

SECRET FUNCTIONS

We want to implement a secret function, i.e., a function which is not completely known to an outside observer. Such is the case for example in cryptology and in access control systems. The input variable x is supposed to be discrete and takes only a limited number of values. The output variable y is a function of x.

$$y=f_\alpha(x) \qquad (2)$$

where α is a fixed but unknown parameter in the range.

$$\alpha=1,\ldots,M \qquad (3)$$

Starting from scratch, an outside observer does not know α, but knows the set of functions f:

$$\{f_\alpha\}_{\alpha=1}^{M} \qquad (4)$$

The problem is now: How does the knowledge of a number of pairs (x,y), satisfying (2), affect the uncertainty about the parameter α? The range of possible α obviously cannot increase if the observer gets to know one more pair (x,y). The problem of the designer of the secret function seems to be maximizing the remaining number of possible α. We will see that this is not good advice. H leads to low uncertainty about y, given x. A better formulation of the problem is therefore: How does the knowledge of a number of pairs (x,y), satisfying (2), affect the uncertainty of y given x?

STRUCTURE OF THE SYSTEM

We may divide the system into a known part, containing the functions (4) and an unknown part containing the parameter α.

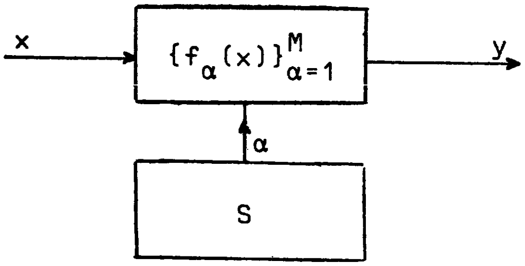


Figure 1—Structure of the secret function

S may be visualized as a store or memory, capable to store the M possible values of α. Hence the capacity, C, of the store must be:

$$C={}^2\log M \text{ bits} \qquad (5)$$

Figure 1 also reflects the structure of the implemented system. The block containing the function f(x) is supposed to be known, while S is a secure memory.

α is regarded as the outcome of stochastic variable A with

$$P[A=\alpha]=1/M \quad \text{for } 1\leq\alpha\leq M \qquad (6)$$

The output is then a stochastic variable Y. n pairs (x,y), satisfying (2), are known to the observer. The uncertainty in Y for a given x is measured as the conditional entropy:

$$H_n\equiv H(Y|Y_1,..Y_n)\equiv$$
$$-\sum_{y}\sum_{y_1}\ldots\sum_{y_n} P(y,y_1,..y_n).{}^2\log P(y|y_1,..y_n) \qquad (7)$$

Note that this entropy is a function of $x,x_1,\ldots x_n$. All entropies are nonnegative. The maximum of H_0 occurs when the possible outcomes of Y are equally likely.

$$\max H_0={}^2\log q \quad \text{when } P(y)=\frac{1}{q} \qquad (8)$$

where q is the number of possible values for the output variable y. If Y is conditionally independent on previous observations $Y_1\ldots Y_n$ then

$$H_n(x)=H_0(x) \text{ as } P(y|y_1,..y_n)=P(y) \qquad (9)$$

Note that H is still a function of x. If we sum H_n over n we obtain the following result:

$$\sum_{n=0}^{N} H_n=-\sum_{y}\cdot\cdot\sum_{y_n} P(y,y_1,..y_n).{}^2\log P(y,y_1..y_n) \qquad (10)$$

If we make N large enough only one or none of the M functions will pass through a given set of points $(x,y),(x_1,y_1),\ldots(x_n,y_n)$. Thus equation (10) reduces to:

$$\lim_{N\to\infty}\sum_{n=0}^{N} H_n=-\sum_{i=1}^{M}\frac{1}{M}{}^2\log\frac{1}{M}=-{}^2\log\frac{1}{M}={}^2\log M \qquad (11)$$

where we have used the probability in equation (6). Finally we combine equations (5) and (11) to:

$$\boxed{\sum_{n=0}^{\infty} H_n = C} \qquad (12)$$

In words: The sum of the entropies at each observation in a sequence of observations equals the capacity (in bits) of the key space. In practice C is the number of bits in the binary key.

Example

In a computer cryptographic system the input, output and key are binary numbers.

In the algorithm proposed by NBS as federal standard,[2] for example, k is 64 bits. (Actually lower because of redundancy in the key.) The output is also a 64-bit word.

Thus the maximum entropy (max H_0 according to (8)) is 64 bits. If H_0 really is 64 bits, then from (12) $H_1, H_2 ..$ is zero. Hence all information about the transformation is given in the first observation. The future outcomes are perfectly predictable:

Now suppose that H_0 is lower than 64 bits. Suppose that we want the system to withstand 8 (eight!) observations before the outside observer can identify the key. Then from (12):

$$\sum_{n=0}^{7} H_n = 64 \text{ bits}$$

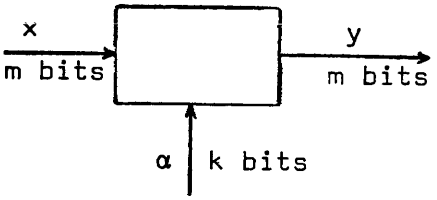


Figure 2—Computer cryptographic transformation

and thus the average of $H_0, .. , H_n$ is 8 bits. With this low entropy it takes only 256 trials (at most!) to make a correct guess of the output for a given input;

The situation is indeed disappointing, from the theoretical point of view!

ONE-WAY FUNCTIONS

We have as yet made no indication as how to compute the estimates of the output or the key from our observations. As we have seen from the above example it may be possible to predict the output and to identify the key after just a few observations. This must be avoided, of course. The only way to stop any effort in this direction is to make it practically impossible to perform the calculations which are theoretically possible.

We refer to Figure 2. The function:

$$y = f(\alpha, x)$$

is calculated by the encryption unit and must be easy to perform. Also:

$$x = f^{-1}(\alpha, y)$$

has to be performed by the decryption unit. On the other hand we have to design the system so that the function

$$\alpha = g(x, y)$$

is practically impossible to compute. This is an example of a one-way function. The algorithm proposed by NBS[2] does indeed have this property. To my knowledge it has successfully resisted every attempt to compute the key, given the input and output.

REFERENCES

1. Shannon, Claude, "Communication Theory of Secrecy Systems," *Bell Syst. Tech. Journal,* Vol. 1949, pp. 656-715.
2. "Computer Data Protection," *Federal Register,* Vol. 40, No. 52, March 17, 1975, pp. 12067-12250.

A secure, national system for electronic funds transfer

by D. KAUFMAN and K. AUERBACH
System Development Corporation
Santa Monica, California

ABSTRACT

This paper presents guidelines for development of a secure national network for electronic funds transfer. Six security principles are developed. These principles, together with certain important networking notions, are utilized to evolve a system level design of a secure localized system for electronic funds transfer. This design is then further defined in order to address the various problems involved when local systems are linked to form a national network. It is concluded that national standards are needed in order to prevent proliferation of incompatible local systems.

INTRODUCTION

As the computerization of bank functions continues its rapid advance, electronic funds transfer is becoming a reality. Independently developed local systems are evolving—but the emergence of a system national in scope is inevitable. Unless planning for security and for operation on a national scale begins now, development of an efficient and secure future system may be impossible.

We believe that a secure, national network for electronic funds transfer (EFTS) can be built with currently available technology. We do not suggest that the monumental task of interconnecting all the various financial institutions in the United States be undertaken, rather we contend that pilot EFTS networks being planned today could and should provide a high degree of security assurance. Furthermore, these pilot systems *could* be built so that as they inevitably grow, proliferate, and interconnect, they can be linked together to form a national network without major impact on either local system structure or local system security and privacy.

EFTS SECURITY PRINCIPLES

As a basis for this discussion of EFTS security principles, several basic assumptions must be made about EFTS schemata. These include:

1. All funds transfer transactions are initiated by a cardholder (possibly assisted by a teller or a merchant) at any of a variety of Point of Sale or Automated Teller devices. These devices are commonly referred to as Remote Service Units (RSUs). Although other transactions not involving a transfer of funds may be handled by an EFTS system, they are not addressed in this discussion to avoid distraction from the major issues addressed.
2. Each bank card has imprinted or recorded on it a personal account number (PAN), institution identification information, and other data such as the expiration date of the card. A cardholder initiating a transaction must supply a value not on the card. This value is called a Personal Identification Number (PIN). The PIN was conceived as an aid in verifying the identity of the user of the card (i.e., the PIN is a password).
3. All funds transfer transactions must be authorized. An authorization, or transaction approval, is based upon a verification of the cardholder's identity and an examination of his account. If the cardholder has supplied the appropriate PIN and if his balance or credit limit is sufficient to allow the transaction, then an authorization is generated. A Host Processing Center (HPC), the computer facility of a financial institution, will typically authorize transactions.
4. Financial institutions may require that the EFTS network provide backup support for the HPC authorization function. For instance, the network may have to provide an alternate site to perform transaction authorizations when the primary HPC is down. Similarly, the EFTS network may be required to log all transactions. These assumptions must be considered in the development of any EFTS network design.

Security Principle #1: The PIN should be known only by the cardholder

It is important to realize that the PIN is potentially a powerful tool for providing EFTS security, and apparently the only currently viable means for positive identification of the cardholder.

The authentication process is important since cards can easily fall into the wrong hands. Cards can, of course, be stolen or lost. Furthermore, any card which can be easily produced can also be easily forged. Electronic funds transfer will provide a powerful incentive to illegally produce and distribute fraudulent bank cards. The identity of cardholders must, therefore, be authenticated.

The PIN, therefore, plays a critical role in EFTS security, and PIN distribution must be carefully controlled. It has been suggested that PINs be stored at the computing facility of the cardholder's financial institution. It may also be desired to store PINs at the network's backup sites. Unfortunately, the greater the distribution of the PIN, the greater is the risk of illegitimate PIN acquisition. For example, if PINs are stored at the bank, they are potentially exposed to dishonest bank employees. And more distressing, if PINs are stored at a backup site, they are potentially exposed to personnel who may not even be under the control of the cardholder's bank.

Only the cardholder need know the PIN if, at the time of issue and within the network, it is transformed by a one-way process to create a unique new value, and if only the transformed version is used to authenticate cardholders. The new value could then be used for cardholder authentication, but the original PIN could not be determined from this new value. Thus, neither the HPC nor the backup sites have access to the original PIN. PIN transformation is discussed in more detail in the system level design portion of this paper.

Security Principle #2: There should be no way to derive the PIN from information on the card

The importance of PIN security to EFTS security is recognized in both the banking and the security communities. Oddly enough, however, many PIN schemes currently being discussed are based upon the notion of deriving the PIN from the information on the card (and primarily from the PAN). Such schemes do reduce the need for PIN storage in the system since PINs can simply be derived when needed, but such schemes risk PIN exposure.

Schemes in which the PIN can be derived from information on the card are inherently weak. Once the algorithm used to convert card information into PINs becomes exposed, any person who obtains the card must be assumed to have obtained the PIN as well. This observation has two important implications in generated PIN systems. First, the secrecy of the PIN depends entirely upon the secrecy of the algorithm used to generate the PIN. Second, the incentive for theft of an algorithm is high, since that algorithm is utilized to generate all PINs for a particular institution's cards. The means for determining such algorithms exists. The algorithms may be exposed by bank personnel who, by the nature of their jobs have access to it, or given sufficient cards with known PINs, it may be possible to synthesize the algorithm. Once the means of deriving PINs is known, production of apparently valid but unauthorized cards is a simple matter. The system level design section of this paper will describe a method of PIN verification which does not require that the PIN be derivable from information on the card.

A rough analogy may be drawn to the security problem of telephone credit accounts. Credit identification numbers are based on the account holder's telephone number, and the time lag between the development of new methods of deriving credit card numbers and the fraudulent use of them has always been short indeed. The potential rewards of defrauding an EFTS system are incalculably greater.

Security Principle #3: Exposure of PINs should be minimized during a transaction

This principle stresses once again the importance of the PIN in EFTS security. A transaction will involve many devices and probably more than one financial institution. PINs should, therefore, be transformed or otherwise protected at the earliest possible stage in the transaction.

Security Principle #4: Sensitive or private transaction data should not be subject to unauthorized exposure

During the course of a transaction, sensitive data passes through a variety of devices and may be transmitted over public communications lines. Not all EFTS devices may be "trustworthy." Communications lines can be easily tapped. Obviously any sensitive data such as the PIN should not be exposed unnecessarily. Furthermore, because privacy statutes are likely to be enacted in the near future, the network must exercise strict control over all personal information involved in transactions. The PAN, for example, may be regarded as private information and not all devices will need to have access to the PAN.

Security Principle #5: Transaction data should not be subject to unauthorized alteration

As transaction processing is performed, alteration of certain data could result in authorization of otherwise illegitimate transactions. For example, transactions may be diverted to the wrong institution or the amount of the transaction might be changed during a transaction, fooling the HPC into authorizing an improper transfer. Protection via an encrypted error detection field is a simple technique to prevent such unauthorized alteration. This technique is detailed later in this paper.

Security Principle #6: All transaction requests and transaction authorizations should be authenticated at their destination

RSUs, where all transaction requests originate, and HPCs, where processing of transactions occur, may be physically remote from one another. However, each must act on information received from the other. It is essential that the identity of the source of information be authenticated by the receiver of the information. An HPC must know that the request it receives actually comes from an RSU and not an outside source, such as a penetrator tapping onto the line. An RSU must know that a transaction authorization actually came from the appropriate HPC. Otherwise a physical transfer of funds or merchandise may occur when the necessary authorization was denied or simply did not take place.

An example will illustrate this point. A grocer rings up a bill for a customer's purchase. The customer wishes to use his card to pay the bill, and wishes to receive an additional $50.00 cash. The grocer enters the transaction request on his RSU and the customer inserts his card and enters his PIN. When the grocer receives an authorization on his RSU, he accepts the transfer as payment and gives the customer $50.00 in cash. A penetrator could have injected a false authorization message somewhere along the line. The grocer would then assume that his account has been credited in the amount of the cash disbursement plus the cost of the groceries, but the "authorization" is fraudulent and the grocer has been cheated. A direct, positive identification of the source of messages in the system must be incorporated to prevent such fraud.

SYSTEM LEVEL DESIGN

The six security principles may now be combined with basic intercomputer network concepts to formulate a general design for a local EFTS system. The following paragraphs describe a design that has the potential to provide a high degree of security assurance.

The design incorporates cryptographic devices. These devices encipher data (i.e., transform data in order to conceal its meaning) and decipher data (i.e., reverse the encipher process in order to render data once again intelligible). Proper use of cryptographic techniques can greatly enhance network security. However, in order to simplify presentation of the design, the system is first presented and analyzed without cryptographic devices. The cryptographic devices are then introduced and discussed in detail. It is important to note, though, that security is an integral part of the entire design.

An EFTS system configuration without cryptographic devices is illustrated in Figure 1. This structure includes four major types of devices or processors.

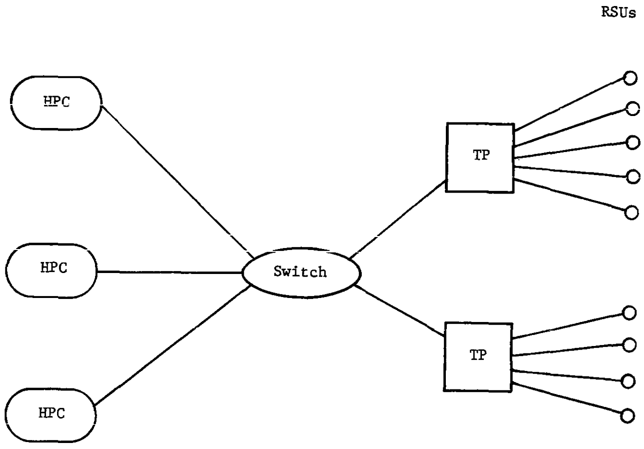


Figure 1—Local EFTS Network (without Cryptographic devices)

Two of these, RSUs and HPCs, were discussed previously. A third type of device, the transaction processor (TP), interfaces RSUs to the rest of the EFTS system, manages funds transfer requests initiated at RSUs, and performs the one-way PIN transformations. The fourth device type, the switch, interconnects HPCs and TPs.

An example (see Figure 2) may help to clarify the function of these devices and the relationship between them. Using the example of the customer at a grocery store, we will assume that the customer maintains his card account at institution X and that the grocer maintains his account at institution Y. The customer desires to use his card to pay his grocery bill of $35.00 and wishes to receive an additional $50.00 cash. The customer inserts his card into the RSU and enters his PIN. The grocer enters a request for a transfer of $85.00 (i.e., $35.00 for the groceries plus $50.00 for the cash the grocer will give the customer) from the customer's card account to the merchant's account. The RSU collects all this information and forwards it to the TP.

The transaction request is then received by the transaction processor. The TP isolates the customer's PIN from the transaction request and derives two new values, PIN′ and PIN″, by performing two successive transformations on the PIN. PIN″ is compared with a set of digits, called cryptographic check digits (CCDs), recorded on the customer's card. If PIN″ is not equal to the CCDs, the PIN is invalid. The funds transfer would not occur and a transaction denial would be sent to the grocer at the RSU. In this example we will assume that the CCDs and PIN″ are equivalent and that transaction processing continues.

The TP then sends a debit request message destined for HPC X, the computer facility of the institution at

RSU	TP	HPC X	HPC Y
1. Grocer enters transaction 2. Customer inserts card into RSU and enters his PIN 3. Transaction data is forwarded to TP	4. Transaction data is received from RSU 5. PIN' and PIN'' are generated 6. PIN'' is checked against CCDs 7. PIN'' check succeeds (If check fails, transaction is aborted at this point.) 8. Debit Approval Message is created and sent to HPC X	9. Debit Request Message is received from TP 10. PIN' is checked 11. PIN' check succeeds (If check fails, rejection is sent to TP and transaction is aborted at this point.) 12. Customer's account balance is checked for sufficiency. (If check fails, rejection is sent to TP and transaction is aborted at this point.)	
	15. Debit Approval Message is received from HPC X 16. Credit Message is created and sent to HPC Y,Transaction Approval Message is created and sent to RSU.	13. Customer's account balance is found sufficient and $85.00 (i.e., the transaction amount] is deducted. 14. Debit Approval Message is created and sent to TP.	
17'. Transaction Approval is received from TP 18'. Customer receives groceries and $50.00 in cash			17. Credit Message is received from TP 18. Grocer's account is credited with $85.00

Figure 2—An EFTS transaction

which the customer has his account. The debit message is addressed to HPC X and transmitted via the switch. It should be noted that the customer's PIN is not transmitted, instead PIN′ is sent along with additional transaction information.

Upon receiving the debit request, HPC X verifies that PIN′ correlates properly with the customer's PAN and that the customer's account balance is sufficient to cover the $85.00 request. If either test were to fail, the debit request would be denied and a debit refusal sent to the TP.

Assuming the debit is approved, HPC X records the debit request, reduces the customer's account balance by $85.00, addresses a debit authorization to the TP and transmits the authorization via the switch.

The TP sends two messages upon receiving the debit authorization. One message is sent to the grocer's RSU, indicating to the grocer that the funds transfer has been approved. The second message is a credit message sent to the HPC Y via the switch. At this point the transaction is completed.

The transaction scenario outlined above demonstrates some basic functions of an EFTS system. Several simplifying assumptions were made to clarify the presentation. Neither backup support for HPCs nor cryptographic devices were included, and logging of transaction data for auditing and accounting was not discussed. Furthermore, message acknowledgments and retransmissions were ignored. Each time a network message is transmitted, an explicit acknowledgment is expected. If an acknowledgment is not received promptly, the message should be retransmitted. Throughout this design presentation we will assume that an acknowledgment/retransmission mechanism exists where appropriate.

In the subsequent, detailed discussion of the local EFTS design, the issues of HPC backup, logging and auditing will be considered. The security of the EFTS system will be analyzed after the full presentation of the system level design.

The switch

The switch interconnects HPCs and TPs. The exact nature of the switch is of no concern here—any switch which is capable of carrying messages to a specified destination in a timely manner is acceptable. In a centralized system the switch may consist of a single message switching computer. On the other hand, the switch may consist of a geographically distributed network of message or packet switching mini-computers. The term "distributed networks" as used in this paper means those networks where messages, or pieces of messages—packets—are carried from source to destination by being relayed from one switching computer to another until the destination is reached. Currently such distributed networks can relay a message across the United States in less than one-half second.

The distributed approach (which is used in the ARPANET) offers many advantages over the centralized approach. Distributed networks have the potential to provide alternate message pathways when one of the switching centers fails. When a centralized switch fails, the entire EFTS system halts. Distributed approaches, besides having a great potential for reliability, may be designed to adaptively route traffic through the various communications paths in order to reduce communications delays.

Unfortunately, distributed systems are not necessarily the most cost effective approach for a local EFTS system. Distributed systems generally require a much higher initial investment than centralized systems. It should be noted, though, that either a centralized or a distributed switch can be incorporated into a local EFTS system without impacting other system components.

Host processing centers

Each HPC is the computer facility for a specific financial institution and as such is subject to the particular policies of that institution. A large and varied population of HPCs now exists. The manner in which accounts are maintained and PINs are handled will undoubtedly vary.

Each HPC must adhere to the message formats and protocols developed for the local EFTS system. All communication between HPCs and TPs must conform to these standards. For instance, HPCs will receive only transformed PINs. The precise manner in which transaction messages are generated, transaction data interpreted, and transformed PINs verified can be determined by each institution.

Functions may be desired in the EFTS system other than those illustrated in the simple transaction scenario presented above. For example, a facility to back up HPCs or to log data on all transactions is likely to be included in most EFTS system requirements. In this EFTS system design these functions are provided by one or more special-purpose HPCs (see Figure 3). The switch need not distinguish between such special-function HPCs and transaction HPCs.

Only TPs and HPCs need to recognize the functions of these special HPCs. It is expected that a TP would transmit a message to the logging HPC at the start and end of each transaction. Similarly, the debit and credit HPCs would transmit log messages to the logging HPC each time they either authorize or refuse a request.

Whenever a primary HPC is not operating, it is expected that TPs would interact with a backup HPC. The backup HPC would partially determine the validity of debit requests based upon information collected from HPCs when they are operating. Transaction information would be stored at the backup HPC until the primary HPC is again operating.

Transaction processor

The TP manages all transactions in the EFTS system. The TP interprets each transaction request received from an RSU. A set of actions is associated with each type of transaction. These actions include a sequence of messages to be sent to HPCs and the RSU initiating the request.

The TP must determine to whom the various transaction messages should be sent. Thus the TP must maintain tables indicating where messages should be routed.

The TP manipulates PINs. Upon receiving a transaction request, the TP creates two transformed PINs, PIN′ and PIN″. Both transformations should be PIN

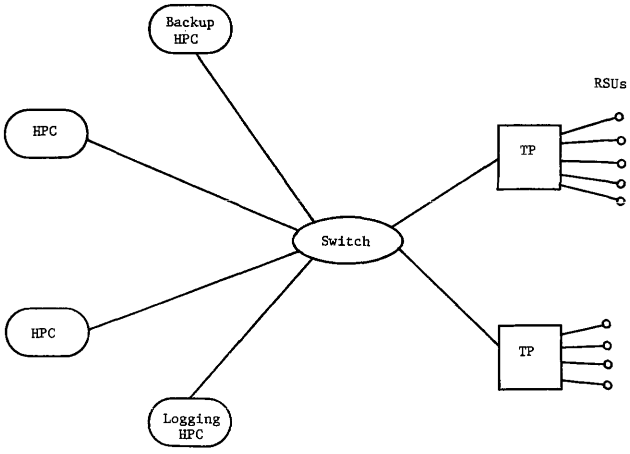


Figure 3—Local EFTS network with special-purpose HPCs

dependent (i.e., they should vary with the value of the PIN) and should be resistant to attempts to determine original PINs from transformed values.

Transformations of this type can be performed in many ways. One such technique employs the NBS standard algorithm for data encryption. This algorithm has two inputs, a text string and a key. The output is a scrambled version of the input text string. The algorithm has the desirable property that even if both a sample input text string and the output are known, the key can only be determined by testing all 76×10^{15} possible keys. (This protects future cyphers from sophisticated penetration attacks.)

The transformation process is illustrated in Figure 4. In this method the PAN is the first text input to the NBS algorithm and the PIN is the key input. The output of the first application of the algorithm is PIN.′ PIN′ is then input to the algorithm as the text and a predetermined but secret value is input as the key. The resulting output is PIN″. Thus both PIN′ and the "secret value" must be known to determine PIN″ and both the PIN and the PAN must be known to determine PIN′. The important security implications of this approach are discussed later.

Cryptographic devices

Two types of cryptographic devices are included in the EFTS system design. These devices are referred to as Network Cryptographic Devices (NCDs) and Serial Cryptographic Devices (SCDs). An EFTS network incorporating cryptographic devices is illustrated in Figures 5 and 6.

The National Bureau of Standards (NBS) Data Encryption Algorithm should be utilized in the SCDs and NCDs. The algorithm has many desirable features for use in such devices (see Reference 1). Furthermore, it is rapidly being accepted as a standard for use in EFTS networks.

SCDs are similar to standard cryptographic devices now available. An SCD protects a single telecommunications line. Multiplexed SCDs can simultaneously handle several such lines. NCDs, on the other hand, are quite unlike anything now produced. NCDs maintain a fully interconnected network. By using a unique key, each NCD can protect the communications path to any other NCD in the network. This technique is described in the following section. It is assumed that an automatic key update mechanism in the NCDs and SCDs changes keys after a given amount of use.

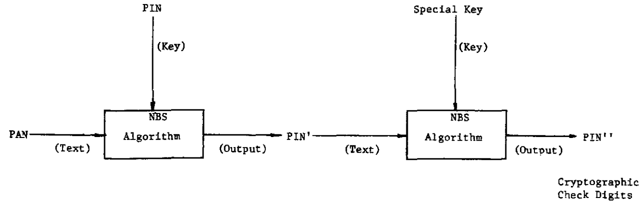


Figure 4—PIN transformation using National Bureau of Standards data encryption algorithm

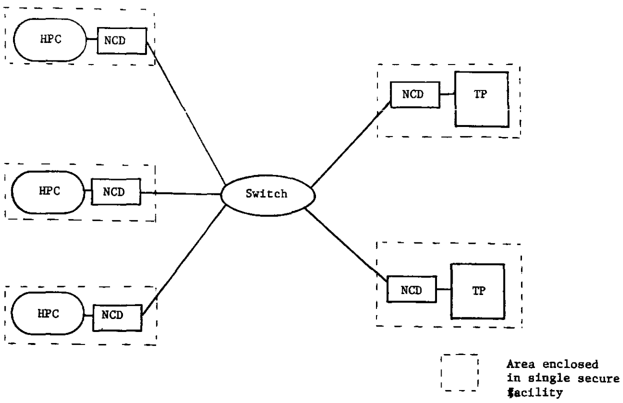


Figure 5—Portion of EFTS network with cryptographic devices

EFTS SYSTEM SECURITY ANALYSIS

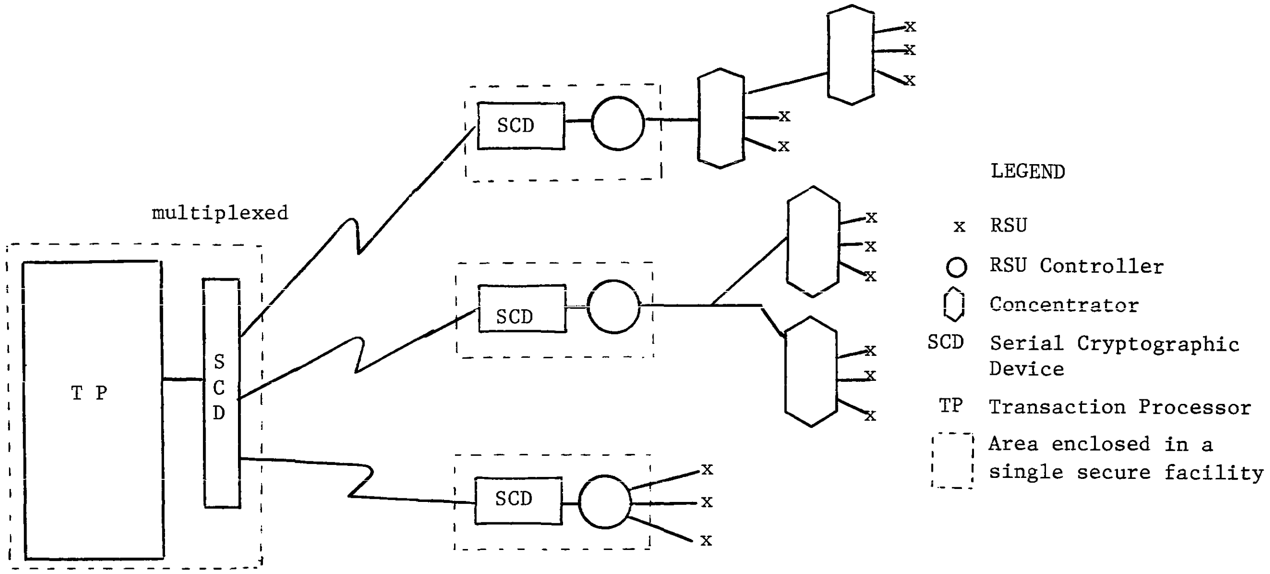
The EFTS system described above should provide substantial security assurance. The following few paragraphs analyze the system's security based upon the six EFTS security principles previously presented.

Security Principle #1: The PIN should be known only by the cardholder

In the system presented above, the PIN is not stored anywhere in the system. All processing beyond the TP is based upon transformed versions of the PIN. HPCs perform authorization checks on transformed PINs only and it is virtually impossible to derive the actual PIN from the transformed PIN.

Security Principle #2: There should be no way to derive the PIN from information on the card

This principle can simply be restated as a system requirement. There is certainly no need in the system presented in this paper to generate PINs from information on the card. The use of cryptographic check digits derived *from* the PIN illustrates that the PIN can be verified without being implicitly exposed on the card.



As shown above, RSUs are not directly connected to TPs or SCDs. RSUs are directly connected to an RSU controller. Several RSUs may be attached to a single controller. This may be accomplished by concentrators, multidrop lines, etc. Communications security between RSU controllers and RSUs is, of necessity, the responsibility of the RSU manufacturer.

Figure 6—Transaction processor—RSU portion of EFTS network

Security Principle #3: Exposure of PINs should be minimized during a transmission

PINs entered at RSUs are in the clear until enciphered by SCDs. PINs are again exposed in TPs. Thereafter, PINs are discarded and only transformed PINS are utilized.

If PINs were transformed at the RSU, only transformed PINs would appear in the network. Unfortunately, many RSUs already exist and none perform the transformation described in the system design. Exposure of the PIN can be reduced further if new RSUs adopt the transformation design proposed herein.

Security Principle #4: Sensitive or private transaction data should not be subject to unauthorized exposure

When data is enciphered, it is considered safe from exposure. Thus, sensitive or private transaction data is safe as it flows between SCDs and as it flows between NCDs. There is, however, a potential weak link between RSUs and their controller. Because RSUs and RSU controllers are built to operate as an integrated unit, the burden of providing communication security between these devices must fall on the manufacturers. Manufacturers should be required to provide this security.

Data is necessarily in clear (non-enciphered) form while in RSUs, RSU controllers, TPs, and HPCs. Consequently these devices will require procedural and physical protection.

Security Principle #5: Transaction data should not be subject to unauthorized alteration

Cryptographic techniques can be used in conjunction with error detection techniques to prevent unauthorized alteration of transaction data. An error detection field is calculated on each message and appended to the message before it is enciphered. Encipherment of data based on the National Bureau of Standards encryption algorithm makes it virtually impossible to alter enciphered data with predictable impact on the data once it is deciphered. Thus, when a message is deciphered and the error detection field recalculated and compared to the value in the message, it is extremely unlikely that any changes made to the enciphered message will not be detected. This technique does not directly prevent unauthorized alteration. It does, however, eliminate any threat due to such alteration since virtually all unauthorized changes to messages can be easily detected. If encipherment is coupled with a procedure for retransmitting messages, incentive for altering data without authorization is eliminated. Thus, SCDs and NCDs combined with appropriate protection of the RSU-RSU controller link prevent unauthorized alteration of transaction data.

Security Principle #6: All transaction requests and transaction authorizations should be authenticated at their destination

NCDs are utilized in this design to authenticate the source of HPC and TP messages. Encipherment and decipherment of messages by NCDs is based upon secret values called keys. An NCD cannot decipher a message unless it knows the key used to encipher the message.

Each NCD will maintain a unique key for communicating with each of the other NCDs in the system. Thus, if TP_1 attached to NCD_1 sends a message to HPC_2 attached to NCD_2, the key used by NCD_1 to encipher the message is known only by NCD_1 and NCD_2. When NCD_2 receives the message, NCD_2 can be assured that the message came from NCD_1. The source of the message which arrives at HPC_2 must therefore be TP_1.

Similarly, SCDs will maintain pairwise-unique keys. This technique provides a means for mutual authentication of TPs and RSU controllers. RSU controllers should be required to have a mechanism for authenticating messages sent between RSU controllers and RSUs. However, RSU to RSU-controller communications are the domain of the manufacturers of these devices.

In this system PINs are known only by cardholders and during a transaction are in the clear only in the TP. A transaction can only be initiated at an RSU since the various cryptographic devices prevent unauthorized insertion of messages into the system. Thus the PIN must be known to initiate a transaction and only a legitimate cardholder can initiate a transaction.

A NATIONAL SYSTEM

The local EFTS system previously described conforms to the six EFTS security principles. That system would provide a high degree of security assurance. By linking several of these local systems it is possible to create a secure national EFTS network. Such a national EFTS network design is illustrated in Figures 7 and 8.

Three major components—a nationwide message switching network, gateways, and NCDs—are needed to link the local systems. The nationwide message switching network carries messages between the local systems. NCDs (like NCDs in the local system) protect messages which flow through the nationwide message switching network. Gateways interface local EFTS systems to the message switching network.

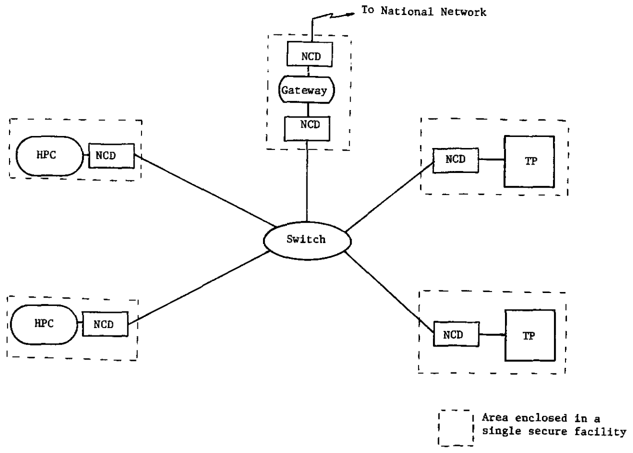


Figure 7—Local EFTS system attached to national network

An example may clarify the function of these internetwork devices. We will assume that TP_1 finds it necessary to send a debit request to HPC_2. We further assume that TP_1 and HPC_2 are not in the same local system.

TP_1, recognizing that HPC_2 is not local, generates a debit request message addressed to HPC_2. That request is enclosed in a message addressed to a local gateway, G_3. The message is transmitted, via the local switch, to G_3. G_3 receives the message and extracts the debit request. The gateway inspects the debit request to determine which local system contains HPC_2. G_3 then encloses the debit request in an inter-

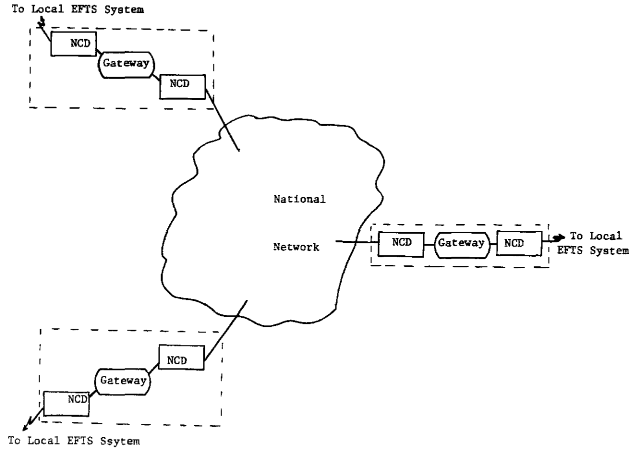


Figure 8—A national EFTS network

network message. The internetwork message is addressed to a gateway, G_4, which is part of the same local system as HPC_2.

The internetwork message, cryptographically protected by NCDs, is routed by the nationwide message switching system to G_4. G_4 receives the internetwork message and extracts the debit request. G_4 then routes the debit request to HPC_2 via the local switch. The resulting debit authorization or denial follows the reverse path from HPC_2 to TP_1.

The national network

Like the local system's switch, the nationwide message switching network may take many forms. Any network capable of carrying messages between gateways in a timely manner is acceptable. The national networks will span large distances and, when compared to local switches, will carry a relatively light EFTS message load. Hence, a distributed shared, public network seems appropriate. Because NCDs protect messages sent through the national network, it is possible to utilize a commercial, value-added network.

Gateways.

A TP views a gateway as a special HPC which represents all HPCs not found in the local system. An HPC views a gateway as a special TP.

The national system design presented above assumed that the local systems to be linked were identical. Unfortunately, such standardization is unlikely. Where little commonality exists between local systems, a national system will be effectively precluded. If the only differences are message formats, gateways can be used to translate the message formats utilized by different local systems. It cannot be stated too strongly—a national EFTS system requires standardization of at least transaction protocols and message information content.

To simplify the format translation task, all messages travelling through the national network will conform to a single, standard protocol and format. If a local system does not conform to the national standard, the gateway to that system must translate messages to and from the national standard. In this way neither HPCs nor TPs are impacted by the differences between the local system and the national system. However, it must be reiterated that gateways can only reformat messages. In all other respects (protocol and information content) local messages must conform to the national standard. The more the local system resembles the national standard, the less complex the gateway becomes.

Security analysis of the national system

The extent to which the national system design adheres to the six EFTS security principles is presented in a two part analysis. First, the protection of the PIN is examined. Second, the protection of transaction communications is examined.

A national network can be built in which all PINs are handled in the same manner as described earlier whether the transaction occurs totally within the local system or whether other local systems are involved. If the national network is built in that way, security principles #1, #2, and #3 are satisfied by the national system design just as they were in the local system design. If, in some local systems a non-standard PIN transformation is used, or if the PIN is not transformed at all, PINs may be exposed. Furthermore, nonstandard PIN handling mechanisms may require ad hoc processing in gateways. Such ad hoc mechanisms would increase cost and decrease security, integrity, and reliability.

The extent to which EFTS security principles #4, #5, and #6 are followed depends entirely upon the local systems. If a local system is built according to the design presented in this paper, then messages are not subject to unauthorized alteration or exposure until they enter a local system not adhering to the security principles. This result occurs because the NCDs of the national system protect against unauthorized exposure and alteration of messages sent between gateways. Furthermore, because the NCDs of the national network prevent misdelivery, a gateway may trust that a message it receives actually originated in the remote local network from which that message appears to have come. If both the source and destination local systems adhere to the security principles, then mutual authentication of the ultimate source and destination of a message is possible.

CONCLUSION

Security must be an integral part of any EFTS system design. Adherence to the six EFTS security principles will provide a high degree of system security. Through the proper use of the NBS algorithm, a system for local electronic funds transfer can be built which conforms to these guidelines for handling PINs and transaction data. Although the devices to implement such systems may not be currently available, the technology to build these devices does exist.

National systems for electronic funds transfer can be created by linking local systems. It is necessary, however, that the local systems be designed to operate as part of a national system—effective and secure after-the-fact linking of heterogeneous local systems may be virtually impossible. National standards *must* be developed to permit interconnection of local systems and to insure a high level of security.

BIBLIOGRAPHY

1. Branstad, D. K., *Encryption Protection in Computer Data Communications*, Fourth Data Communications Symposium, Quebec City, Canada, October 1975.
2. Branstad, D. K., "Security Aspects of Computer Networks," Paper Number 73-427, *AIAA Computer Network Conference*, Huntsville, Alabama, April 1973.
3. Cerf, V. and R. E. Kahn, "A Protocol for Packet Network Intercommunication," *IEEE Transactions on Communications*, Vol. COM-22, No. 5, May 1974.
4. Ferdman, M. D., D. W. Lambert and D. W. Snow, *Security Aspects of Bank Card Systems*, MITRE Technical Report MTR-2971, Vols. 1 and 2 and Executive Summary, September 1975.
5. National Bureau of Standards Data Encryption Algorithm, *Federal Register*, March 1975.
6. *Security and Reliability in Electronic Systems for Payments*, Study Group on Electronic Systems for International Payments of the Group of Computer Experts of the Central Banks of the Group of Ten Countries and Switzerland, April 1975.

Design considerations for electronic funds transfer switch system development

by JOSEPH P. MAZZETTI
Technology Management Incorporated
Washington, D.C.

ABSTRACT

This paper reviews the EFTS switch concept and outlines some of the major design considerations involved in its implementation. The EFTS switch permits financial institutions to share customer terminal devices (for example, point-of-sale terminals in a supermarket) by transmitting messages generated at the terminal to the financial institutions holding the customer (and merchant) account. The switch, in addition to message routing, must maintain information for settlement among the financial institutions involved, and generate accounting, audit trial and operational reports. Specifically addressed in the paper are: the financial transactions and terminal devices involved; switch message processing and accounting functions; hardware, software, and network components and alternatives; and security and backup considerations. The material presented is based on EFTS project work performed by Technology Management Incorporated (TMI) for the Federal Home Loan Bank System.

INTRODUCTION

An Electronic Funds Transfer System involving shared terminals *—for example, financial institutions sharing point-of-sale devices in a supermarket or an automated teller machine in a shopping center—requires a computerized switch to route messages from a given device to the computer system (host computer) servicing a particular financial institution. A typical switch configuration servicing two institutions is shown in Figure 1. In addition to message routing, the EFTS switch is designed to generate information for interinstitutional settlement and to maintain accounting and audit trials.

The EFTS switch design emphasizes throughput performance while at the same time providing a high degree of message protection and system integrity. Further, a switch must be capable of handling a multiplicity of host computers with different operating characteristics, have the capability of dealing with varied terminal types, and communicate with other switch networks.

The EFTS switch will be a key element in the future of Electronic Funds Transfer because neither the public nor the merchant community will tolerate a profusion of competing terminals and because of the higher costs associated with independent facilities. The technology for switch development is well within the current state-of-the-art; what remains is the application of that technology to real-world situations. Further, a number of alternative switch and network design approaches exist providing EFTS planners in a given area the ability to configure the switch(es) which best meets the geographical, demographic, and market demand characteristics of that area.

This paper treats basic switch concepts and outlines

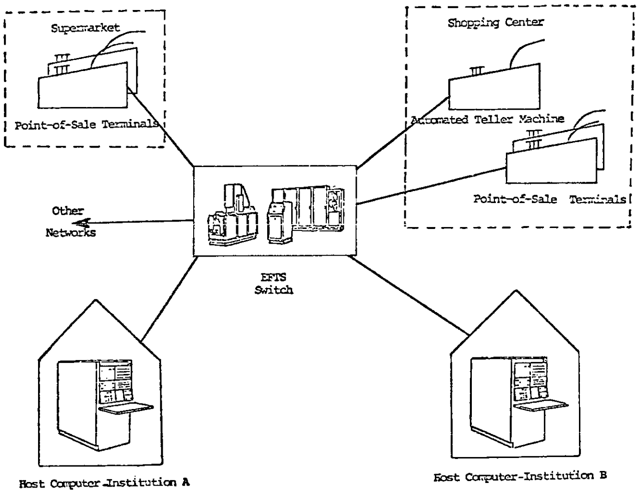


Figure 1—EFTS switch overview

* Commonly called Customer Bank Communications Terminals (CBCTs) by commercial banks or Remote Service Units (RSUs) by thrift institutions.

the major design considerations involved in switch development. It is based on EFTS project work performed by Technology Management Incorporated (TMI) for the Federal Home Loan Bank System.

OPERATIONAL CONCEPT

A range of plastic card transactions initiated by depositors will be processed by the switch, including:

- *cash withdrawal*—direct withdrawal from a checking or savings account;
- *check cashing/guarantee*—placing a "hold" on the customer's account in the amount of the transaction;
- *deposit*—placing funds on deposit to a customer's account;
- *funds transfer*—transfer of funds from one account to another;
- *payments*—direct submission of cash or check for loan payment (e.g., mortgage loan); and
- *balance inquiry*—determining the existing balance in an account.

Other switch-based services could include: interface with Automatic Clearing Houses (ACH); interconnection with Credit and Debit card networks; possible linkage with the FEDWIRE, BANKWIRE, and other networks; and other depositor-based services.

These transactions are generated from several types of terminal devices as shown in Figure 2, including automated teller machines and merchant-operated terminals. The automated teller machines (ATMs) dispense cash, accept deposits, accept payments, and transfer funds between customer accounts, and are activated by a combination of a plastic card and push buttons. They typically limit the total number of withdrawals a customer can make through controls built into the system (on-line operations) and on the customer's plastic card (off-line operations). The merchant-operated terminals are those devices normally operated by merchant personnel in a business establishment such as a supermarket or retail store. They are of two general types:

- point-of-sale terminals used to collect sales, inventory, and other data about a sales transaction in addition to performing some financial transactions. These include: Electronic Cash Registers (ECR) and Point-of-Sale (POS) terminals; and
- terminals located in a business establishment for the sole purpose of performing financial transactions (deposits, withdrawals, etc.). These are often called point-of-business terminals.

These terminals usually include the following components: plastic card, magnetic stripe reader; numeric keyboard and function keys; display; printer or imprinter; journal tape unit (ECR); and usually a terminal controller to service multiple devices.

The flow of data for a typical transaction is shown in Figure 3. In this example, a customer makes a cash withdrawal at a supermarket; the customer's and merchant's accounts are maintained at different financial institutions.

The customer submits his plastic card to the financial service window at the supermarket. The clerk enters the customer's plastic card along with merchant data (either by a second plastic card or key entry). The customer enters his Personal Identification Number (PIN) through a separate numeric key pad. The transaction data is then transmitted to the switch

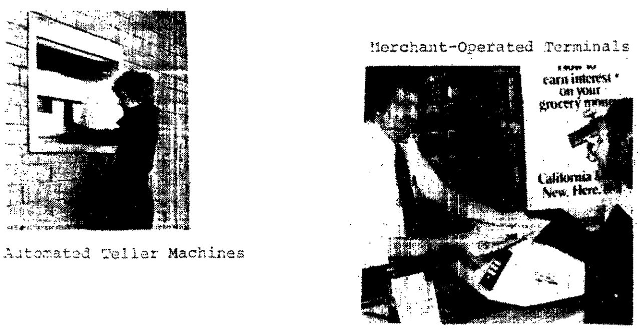
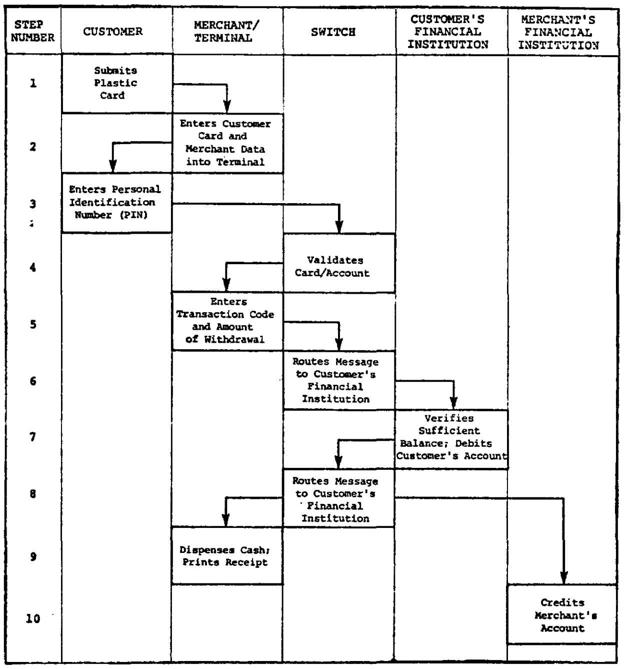


Figure 2—Terminal device types



STEP NUMBER	CUSTOMER	MERCHANT/ TERMINAL	SWITCH	CUSTOMER'S FINANCIAL INSTITUTION	MERCHANT'S FINANCIAL INSTITUTION
1	Submits Plastic Card				
2		Enters Customer Card and Merchant Data into Terminal			
3	Enters Personal Identification Number (PIN)				
4			Validates Card/Account		
5		Enters Transaction Code and Amount of Withdrawal			
6			Routes Message to Customer's Financial Institution		
7				Verifies Sufficient Balance; Debits Customer's Account	
8			Routes Message to Customer's Financial Institution		
9		Dispenses Cash; Prints Receipt			
10					Credits Merchant's Account

Figure 3—Transaction data flow-cash withdrawal

where card and account validation and security is performed through algorithmic procedures and reference to a negative file containing stolen or counterfeit card information. Assuming the validation checks are passed, an acknowledgement is sent back to the terminal. The clerk enters the transaction code and the amount, and the message is then transmitted to the switch. The switch routes the message to the customer's financial institution processor which verifies that a sufficient balance exists to cover the withdrawal and debits the customer's account. A positive acknowledgement of the action is sent to the switch which, in turn, transmits the acknowledgement to the terminal (where a receipt is printed) and also transmits a message to the merchant's financial institution processor where the merchant's account is credited.

The processing performed in each of these steps in the case of a cash withdrawal transaction may vary in alternative switch designs, depending on the division of functions between the switch and financial institution processors.

MAJOR DESIGN CONSIDERATIONS

This section describes in some detail the major design considerations involved in EFTS switch development, including: switch functions, hardware and software components, network alternatives for terminal support and degree of centralization, backup, security, and message standardization.

Switch functions

The EFTS switch must be designed with maximum emphasis on throughput performance (i.e., low message residency time as in an inquiry/response communications network) and also on the ability to provide a high level of system integrity and message protection as can be found in a store-and-forward message processing system. General switch design considerations include:

- ability to support the processing of plastic card-based financial transactions described above;
- ability to provide interconnection with a variety of host processors and terminal devices;
- modular system architecture to accommodate evolutionary transaction volume growth so that no major structural changes are required as greater levels of volume are reached;
- flexible hardware and software design to permit new terminal devices to be added in the future; and
- a high degree of system reliability.

The specific functions to be performed by the switch are listed in Figure 4 and discussed below.

- MESSAGE PROCESSING
- SETTLEMENT
- REPORT GENERATION
- AUDIT TRAIL MAINTENANCE

Figure 4—Switch functions

Message processing

The elements of common processing for each transaction message include:

- *message receipt*—receipt of messages transmitted from terminals and terminal subsystems (i.e., controllers and concentrators), host processors, and other switches;
- *message validation*—verification of the format and content of the message received;
- *account verification and security*—verification of customer and merchant account numbers;
- *message logging*—writing of all messages processed by the switch to a historical file (usually magnetic tape) for audit trail purposes;
- *settlement*—posting accounting data to an interinstitutional settlement file;
- *message reformatting*—reformatting of message as necessary to achieve the appropriate terminal/host processor interface; and
- *routing*—directing the output message to the appropriate endpoint.

Settlement

The settlement process provides for transferring funds between financial institutions to cover the value of interinstitutional transactions originated by the customers of those institutions. This is normally accomplished by moving the funds between the clearing accounts held by a common financial intermediary such as a Federal Reserve Bank or a commercial bank.

Settlement can be handled in a number of ways, depending primarily on the relationship between the entity that operates the switch facility and the financial intermediary which holds the clearing accounts. If the switch is operated by the financial intermediary which holds the clearing accounts, a continuous clearing process can be implemented. This allows instantaneous clearing, and each transfer can be applied, as it occurs, to the clearing accounts of both the sending and receiving institutions. Periodic cutoffs could provide for reporting, reconciliation, and analysis.

In circumstances where continuous settlement is impractical or impossible, batch settlement can be used. This involves the accumulation of transaction data between cutoffs for subsequent posting to a clearing account. A single settlement entry can then be made for each participating institution. That entry represents the net of all deposits and withdrawals initiated by that institution's cardholders through its own terminals.

Regardless of the settlement process used, the switch operation will have to provide detailed transaction reports to enable the participating institutions to reconcile their settlement accounts.

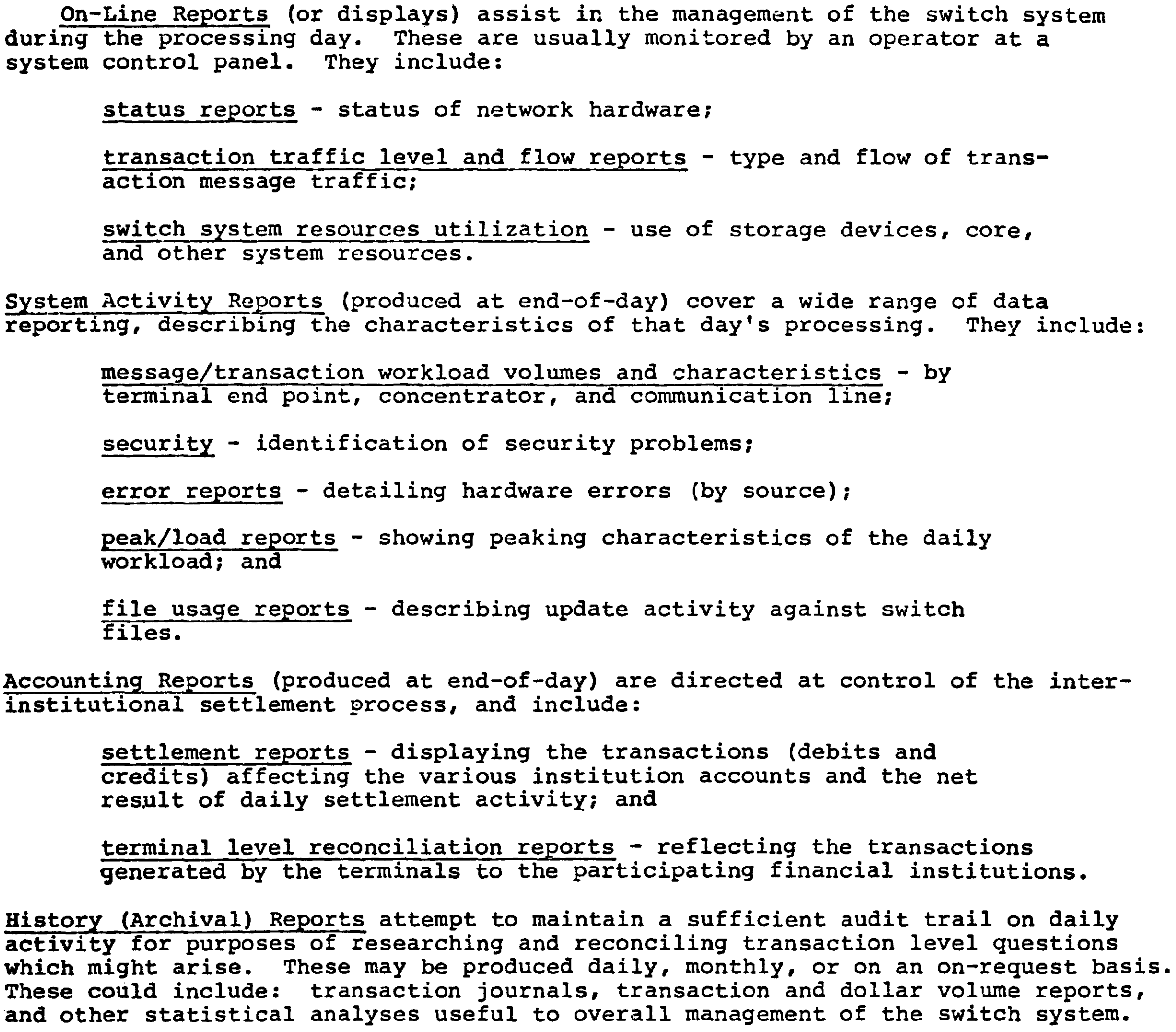
In addition to accomplishing the settlement function, the switch should provide on-line availability of net position data (among institutions) which is a continually updated balance or a beginning balance and net debits and credits. Warning levels can be established at which an institution would be notified of a need to transfer additional funds into the clearing account.

Report generation

The switch must provide data for the production of reports of four types: on-line reports, system activity reports, accounting reports, and history reports. Definitions of these reports are given in Figure 5.

Audit trail maintenance

In order to ensure the ability to restore system operation following an outage, identify patterns of terminal or network use for possible security violations, and assist in reconciliation of switch/host processor/ter-



On-Line Reports (or displays) assist in the management of the switch system during the processing day. These are usually monitored by an operator at a system control panel. They include:

status reports - status of network hardware;

transaction traffic level and flow reports - type and flow of transaction message traffic;

switch system resources utilization - use of storage devices, core, and other system resources.

System Activity Reports (produced at end-of-day) cover a wide range of data reporting, describing the characteristics of that day's processing. They include:

message/transaction workload volumes and characteristics - by terminal end point, concentrator, and communication line;

security - identification of security problems;

error reports - detailing hardware errors (by source);

peak/load reports - showing peaking characteristics of the daily workload; and

file usage reports - describing update activity against switch files.

Accounting Reports (produced at end-of-day) are directed at control of the inter-institutional settlement process, and include:

settlement reports - displaying the transactions (debits and credits) affecting the various institution accounts and the net result of daily settlement activity; and

terminal level reconciliation reports - reflecting the transactions generated by the terminals to the participating financial institutions.

History (Archival) Reports attempt to maintain a sufficient audit trail on daily activity for purposes of researching and reconciling transaction level questions which might arise. These may be produced daily, monthly, or on an on-request basis. These could include: transaction journals, transaction and dollar volume reports, and other statistical analyses useful to overall management of the switch system.

Figure 5—Switch reports

minal interactions, an audit trail must be maintained of each transaction processed by the switch.

All messages received by the switch must be recorded on magnetic tape or some other suitable device. The tape record should include—in addition to the incoming data—a time and date stamp, a sequential reference number, and the identification of the line and terminal or other device from which the data originated. In addition, the record should contain information relating to the disposition of the transaction, such as routing, format revisions, and error codes.

These transaction records will have to be retained for a period of time commensurate with the need for reprocessing, trouble-shooting, and analysis relating to system integrity and security matters.

The hardcopy could be in the form of microfilm, microfiche, printed listings, or another suitable media.

Hardware and software components

The major hardware and software components required for EFTS switch operation are described below. The mix of components will differ for each switch installation based on functions to be performed, transaction workload and performance requirements.

Hardware components will include:

- *Processors*—Several classes of computers could feasibly be used in the switch configuration, including microprocessors, minicomputers, and medium-scale processors. The medium-scale processors would have the capability to perform other financial processing functions, together with the switch functions (integrated switch). For purposes of switch reliability, multiple processors are suggested.
- *Processor Memory*—The range of processor memory sizes will vary according to the class of processor used and the functions to be performed by the switch.
- *Communications Adapters*—These are utilized for communications line control and multiplexing.
- *Input/Output Controllers and Peripheral Devices*—The specific components necessary for switch configurations include:
    - tape drives—for transaction audit log and interfacing with other outside systems;
    - disk—for program storage, switch file residence, and working storage for message processing operations;
    - printer—for production of accounting and activity reports;
    - system console—display-type device for monitoring system operations; and
    - communication "front-ends"—for network control and message handling.

All software required for the EFTS switch can be categorized as either environmental software or applications programs. Environmental software is the manufacturer-supplied packages, including:

- *Operating System*—supports the effective, shared utilization of system hardware and software resources;
- *Transaction/Message Handling Software*—supports communication of messages from terminal to computer, computer to terminal, and from one computer to another;
- *Network Support Software*—descriptive, table-driven language used by a communications processor in controlling network configuration and operation;
- *File Maintenance Software*—provides data base access and update capability;
- *Recovery Software*—provides an automated means for re-establishing operations in the event of system failure (e.g., transmit messages previously received but not yet forwarded); and
- *System Utilities*—includes language processors, data manipulation routines, and other system support software.

The application programs are usually unique to a given switch situation and include those needed to perform the functions described earlier, i.e., message processing, settlement, report generation, and audit trail maintenance.

Network alternatives—terminal support

Several network configurations are possible to provide terminal support for the EFTS switch. Switch performance, economics of operation, initial development cost, maintenance costs, minimization of the number of communication lines and distance, and other factors should be considered in determining the most appropriate configuration in any given situation. Three possible alternatives are depicted in Figure 6 and are discussed below.

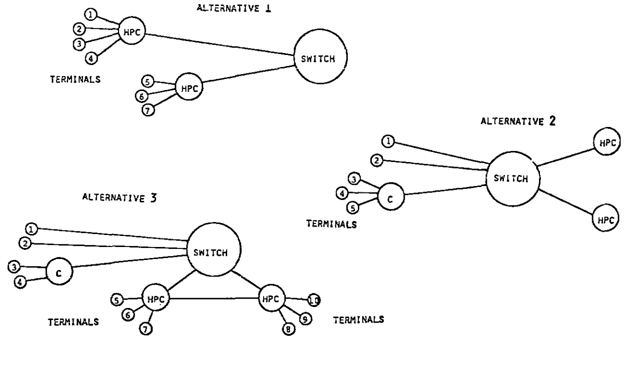


Figure 6—Network alternatives—Terminal support

- *Alternative 1—Computer/Computer Switch*—This approach would require that *all* terminals be connected to host processing centers with those host processors, in turn, connected to the EFTS switch. The host processors would have the responsibility for terminal and network control functions, including: communication line disciplines, error recovery, polling and addressing, and message format translation. Messages would be presented to the switch in a standard format.
- *Alternative 2—Terminal/Computer Switch*—This approach is at the opposite extreme from Alternative 1 requiring that *all* terminals be connected either directly or through a series of concentrators to the switch, and through the switch to the host processing centers. The switch, of course, would also handle interinstitutional settlement. In this case, the switch has complete terminal and network control including message formats, interface specifications, and network configurations.
- *Alternative 3—"Hybrid" Approach*—This approach would provide for:
  - Some terminals connected directly to the switch;
  - Some terminals connected to host processing centers and the host processing centers connected to the switch; and
  - Some computer centers connected directly to each other over high-speed lines (by-passing the switch for certain types of message traffic).

In the short term, Alternative 1 appears to be the most expedient because it reduces the complexity of interfacing various terminal devices (and related standard setting) and reduces the *switch* software and communications costs involved. Alternative 2 presents, in the longer term, the most likely point of evolution because it provides the centralized control and flexibility needed for network expansion. With Alternative 3, several key control questions exist relating to the integrity of the network in the event of failure.

Network alternatives—centralized vs. distributed switch

Figure 7 presents two alternative network design philosophies which are applicable: the centralized switch and the distributed switch.

- The *centralized switch* approach uses one or more computers as the heart of the network processing all transactions. For network configuration (a), transactions generated at terminals are transmitted to the switch and routed by the switch to the appropriate host processing center (and settlement accounting is performed).

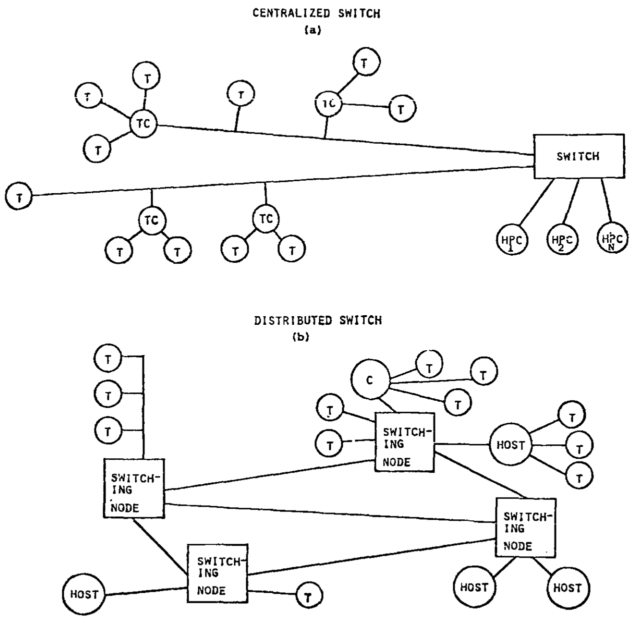


Figure 7—Network alternatives—Centralized vs. distributed switch

- In the *distributed switch* (b), each switching node is designed to be self-contained, operating independently of the other nodes in the network. Normally, at least two communication paths are possible to reach each switch node. Further, if the network is constructed such that specific endpoints (such as a host processor) have connections to two nodes (primary and backup), the functions of that endpoint would be available in the event of failure of a given node.

Backup considerations

The reliability of the switch network depends largely on the switch design itself. For most switch applications, the need for multiple processors is indicated. For the centralized switch, several backup approaches are possible as discussed below for a two processor example:

- *Foreground/Background Processors*—There are two processors; one CPU operates on-line performing all switch functions, while a backup CPU (with access to all peripheral devices) is available to perform off-line functions until the foreground processor goes down.
- *Load Sharing*—There are two processors, each of which services half of the network workload under normal conditions. In the event that one pro-

cessor fails, the second processor takes over its workload.
- *Hot Standby*—There are two processors; each processor is capable of supporting the network workload and processes each transaction separately. In the event that the main processor fails, the standby computer takes over network operation.

The distributed switch network contains inherent backup in that the overall workload is segmented; thus, only a portion is vulnerable at any particular time. Further, alternate transaction routing among nodes in the network also increases the network reliability.

Another factor affecting the reliability and integrity of the switch is the segregation (physical and logical) of switch functions. This segregation is possible at several levels:

- *Software Segmentation*—Use of modular software design to maintain the processing integrity of each of the functional segments described earlier;
- *Separate Processors*—Use of dedicated processors to perform one or more of the functions indicated; and
- *Sharing*—Assignment of one or more processors in the network to perform given functions; e.g., assigning a host processor to maintain all settlement information.

This segregation can affect the operational efficiency of the switch, its reliability, and its ability to recover in the event of an outage.

Switch security

The process of transferring funds between customer accounts and from one financial institution to another via depositor-activated or merchant-operated terminals is susceptible to many types of fraud. Switch designers must build security protection into the basic design. Figure 8 highlights some of the major areas of "vulnerability". Examples of possible fraud are: counterfeit, changed or duplicated cards; wire tapping; tampering with terminal devices; adjusting switch or HPC programs; modification of switch or HPC files.

Commonly considered approaches to achieving adequate security include:

- rendering the card counterfeit-proof and tamperproof by embedding various materials on or within the card during manufacture;
- encrypting procedures, such as enciphering the characters on the magnetic stripe, and enciphering all file data;
- physical security procedures for switch and host processing center facilities; and
- systems for signature verification, voiceprint, and fingerprint identification.

- PLASTIC CARD
- TERMINAL
- MAN-MACHINE INTERFACE
- COMMUNICATIONS INTERFACE
- SWITCH SOFTWARE
- SWITCH FILES
- SWITCH PERSONNEL AND FACILITIES
- HOST PROCESSOR (FILES, PROGRAMS, PERSONNEL, FACILITIES)

Figure 8—Security considerations

Message standardization

The issue of message standardization has a direct bearing on the cost of development and on-going maintenance of an EFTS switch, its operational performance, and the degree of cooperation among those institutions participating in the use of the switch. Consider two opposite approaches:

- *Standard formats*—This approach would require that all messages (i.e., terminal/switch and switch/host processor) be standardized. This would provide considerable ease of administration and control, as well as a lower base of development and operating costs. On the other hand, practical experience has proven that agreement on standards is very difficult to obtain and, during an interim period, the changeover to standard formats would require maintenance of dual formats with the attendant operational and control problems.
- *No standard formats*—This approach would require that the switch have available a set of terminal-specific and host processor-specific programs which are used whenever a given terminal (or host processor) interaction is initiated. It would permit any new terminal or host processor to be brought "on stream" with minimum switch-imposed restrictions or modifications. It would, however, involve a large investment in software devel-

opment and maintenance and could increase overall transaction processing time.

Between the two extremes, there exists the possibility of standardizing portions of message processing: such as between the terminal and the switch or between the host processor and the switch. Over the long term, as EFTS networks expand, standardization of message formats will become more important. As the number of host processors and other switch nodes increases, control and maintenance problems will also increase.

RECAP

The technology to support EFTS switch development in a range of alternative design configurations as described above is generally available today. EFTS development work is currently under way in pilot projects throughout the country and hardware and software vendors offer a variety of relevant product lines.

More detailed investigation is still required in several areas, specifically:

- *Security*—There is a need to determine the degree of security (over terminal devices, communications network, and switch operation) which is required to ensure against violations of privacy and fraud; and how that level of security can be delivered at a price which does not render switch operation prohibitively expensive.
- *Standards*—A lack of standardization will, in the long run, lead both to unnecessary duplication of effort and the intercommunication problems both within and outside of a given switch area. Standards will be required in message formats, communication line protocols, terminal device characteristics, and error detection and correction mechanisms.
- *Other Considerations*—Further definition is required in areas relating to settlement and audit trail including: float implications of switch operation (e.g., direct debiting of an account versus value dating of the transaction for future debiting); clearing arrangements for the financial institutions participating in the switch; and historical audit trail requirements for transaction, terminal, and store level reconciliation of customer and merchant accounts.

Are computers ready for the checkless society?

by FRANK BACKMAN
IBM Design Center
Gaithersburg, Maryland

ABSTRACT

Once the furor over the legal and regulatory issues of electronic funds transfer dies down, the data processing industry will find several complications in the development of a systems architecture suitable for implementing a nationwide system. Apart from interface difficulties presented by the equipment and software designs of competing manufacturers, there are complications to the control of such a system that appear to merit the attention of the data processing industry. The purpose of this paper is to stimulate the exchange of opinions on how the industry should cope with differences in end-user protocols, different routing techniques, different methods of controlling terminal devices, the value of various communication line disciplines, and the overall management of a nationwide network.

BACKGROUND

For the past several years the move toward an electronic funds transfer system has been gaining momentum. The Federal Reserve System already transfers vast sums of money electrically; banking terminals are popping up in supermarkets and airports; the battle lines between the thrift institutions and the commercial banks have been drawn; and reliable digital communications are becoming available at an unprecedented rate. Even the most strident slow-payers can be accommodated in the systems being proposed. It seems inevitable that the flood of paper that is engulfing the financial industry is about to be abated by a worldwide network of digital computers and terminals. After all, stored-program digital devices can do anything!

The problem is that they don't; and there seems to be a labyrinth of data processing issues that will become critical, once the furor over legal and regulatory rules dies down.

To address these data processing issues, the Federal Home Loan Bank Board, in March 1975, invited industry to suggest an electronic funds transfer system that could evolve from the regional needs of a mythical metropolitan area called Middletown. The Federal Home Loan Bank Board provided a scenario that illustrated some of the problems being faced by the architects of an electronic funds transfer system today. The scenario carried the evolution of the EFTS system through several stages in which Savings and Loan Associations entered agreements with local merchants to handle debit and credit card transactions, installed banking terminals at shopping centers, added new functions, and included new S & L's into the evolving system. The scenario, which included enterprises with such picturesque names as Korn Krib Supermarkets and Marshall Prairie Department Stores, was made even more vivid by the technical difficulties posed at each stage of the evolution.

Several computer and terminal manufacturers' equipment was involved, and an equally complicated mixture of transaction types, communication line disciplines and business practices were represented. Respondents to the FHLBB invitation were not requested to submit the detailed design of a system that would track the evolution of the Middletown scenario through its various stages, but rather to postulate a system that would solve similar problems scaled for a metropolitan area with a population of from two million to five million people.

The purpose of this paper is to identify some of these problems that are of a data processing nature (ignoring legal and regulatory issues), to suggest promising directions, and to stimulate discussion among the architects of the future Electronic Funds Transfer System.

ASSUMPTIONS

The arrangement of the institutions on the topographic maps at the end of this paper has been synthesized from the demographic information in the scenario developed by the Federal Home Loan Bank Board and illustrates several points, other than purely technical considerations, that will affect the evolution of an electronic funds transfer switch. The evolution shown for the Middletown environment is intended to represent only a *plausible* evolution through the various stages under the assumption that no consortium is formed to bypass stages. In actual practice, the decision to install a regional switch may be reached at any time, so that some of the technical problems in the expanded scenario need never be addressed.

In the Middletown environment, we have assumed that some transactions may cross a state line. Although this consideration is difficult to cope with in the design of a generalized system, it presents a real-world problem because of the variety of inter- and intra-state communication line tariffs and because of state banking laws. This is a non-technical consideration that will influence the topology of the network.

The location of the thrift institution servicers in the downtown location suggests the possibility of collocation which would, if implemented, have a profound impact on the design of the communication system. We have, however, assumed that the thrift institution servicers are physically separated from one another and that communications lines are required.

The geographic layout of Middletown (as shown in stage 1 and 2a), suggests that these stages would be implemented by a combination of point-to-point and multipoint lines rather than by point-to-point lines only as was illustrated in the original scenario. In addition, reliability considerations may cause First Federal to install two separate multipoint lines in each of its branches. This would permit each branch to tie half its terminals to each line so that business could continue normally in the event of a single line failure. The practicality of doing this is dictated by the prices and the line routing of the local common carriers and is another illustration of a non-technical consideration that will influence network topology.

In the later stages of the evolution, we have assumed that the network would evolve as in the scenario with the thrift institutions and merchants entering agreements with one another at each stage. In order to preserve the staging of the scenario we have also assumed that the rival thrift institution servicers would somehow be able to overcome the technical problems of "looking like one another's terminals" and would accommodate one another's customers by developing and installing the software necessary to cope with the different transaction formats used by the various thrift institutions. This latter assumption, of course, is optimistic, but illustrates a natural tendency toward the evolution of a hybrid network. We have however, assumed that at some point (illustrated in stage 3D) a regional switch is installed and that S & L's subscribe to it through their servicers.

The remainder of this paper describes some key elements of network design and suggests that, once non-technical problems have been overcome and adequate computational "horsepower" has been planned for, the network design is dominated by data processing software and procedural considerations rather than by communications considerations.

NETWORK DESIGN CONSIDERATIONS

Figure 1 shows a simple hierarchical structure for an EFTS network. In practice, the network will be more complex than that shown, since it will frequently be advantageous for terminals to access the switch directly rather than via the computers serving the thrift institutions. (In the scenario, for example, Second Federal may decide to attach their teller terminals and their Merchant Operated Terminals at Fast Food directly to the switch, rather than through the FHLB data center 250 miles away. This decision would save 250 miles of communication line at the expense of increased switching load that would be generated by any "on us" transactions. This situation, can be expected in the actual EFTS network. (The decision will also be influenced by the compatibility of Second Federal's terminals with the switch.) Furthermore, traffic considerations will undoubtedly justify additional intercommunication links among terminals. Nevertheless, even the tree structure of Figure 1 illustrates five key considerations that will dominate the design of the EFTS network. These five considerations, listed on Figure 1 form the basis for IBM's Systems Network Architecture (SNA), and are being addressed by other implementers of specialized computer networks. As yet, however, there is no set of widely accepted standards for computer networking. No matter what standards are adopted by the Federal Home Loan Banking System, these management problems must be addressed

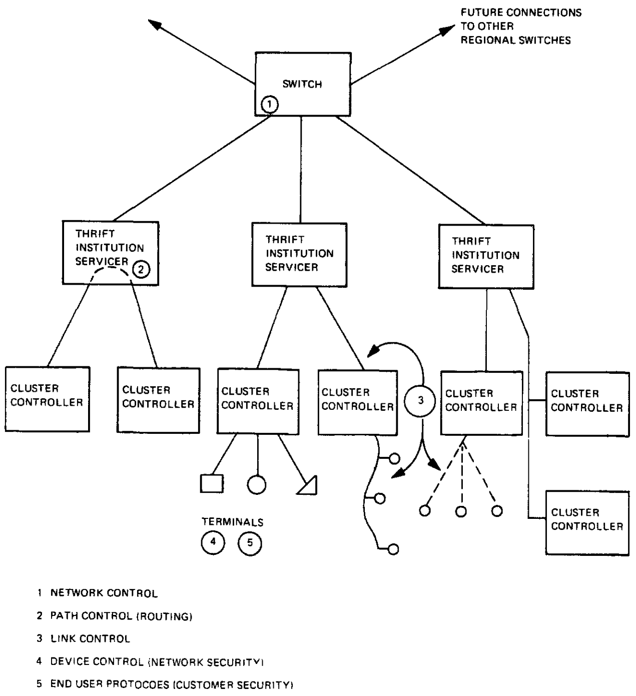

FUTURE CONNECTIONS TO OTHER REGIONAL SWITCHES
SWITCH
1
THRIFT INSTITUTION SERVICER
2
THRIFT INSTITUTION SERVICER
THRIFT INSTITUTION SERVICER
CLUSTER CONTROLLER
CLUSTER CONTROLLER
CLUSTER CONTROLLER
CLUSTER CONTROLLER
3
CLUSTER CONTROLLER
CLUSTER CONTROLLER
CLUSTER CONTROLLER
TERMINALS
4
5
1 NETWORK CONTROL
2 PATH CONTROL (ROUTING)
3 LINK CONTROL
4 DEVICE CONTROL (NETWORK SECURITY)
5 END USER PROTOCOES (CUSTOMER SECURITY)

Figure 1—A simple topography

in the design of the EFTS network and the participating thrift institution servicers must modify their systems to conform to the EFTS network rules.

1. *Network Control*—In order to allow the future interconnection of EFTS switches, some form of distributed control must be used. Control includes responsibility for the equipment configuration in the network subordinate to the control locations, the loading of the communication programs in the processors and cluster controllers of the thrift institution servicers, the reaction to outages, and the activation and deactivation of equipment in response to changing workloads. The Systems Network Architecture, as currently implemented by IBM, concentrates control at the highest level of a tree-structured network and distributes responsibility for its execution to subordinate elements. In the EFTS, control would reside at the switch with the bulk of the responsibility delegated to the thrift institution servicers.

   Special arrangements must be made to crosstell status information between thrift institution servicers when direct links are installed. (In the scenario, for example, it is conceivable that a direct link could be established between two of the data centers, bypassing the switch, and dedicated to high volume transactions between these servicers. It is important that the overall network design not preclude this sort of connection if the member banks feel that it is to their advantage. This situation is invisible to the switch but requires the development of special protocols by the servicers. Further complications to network control are discussed later and must be resolved by the standards adopted by the Federal Home Loan Banking System.
2. *Path Control,* or routing, is the second key consideration in the EFTS network design. Many store-and-forward systems employ a fixed routing technique (with a small number of alternatives) that is specified when the system is initially program-loaded. If the EFTS is to be permitted to evolve into a national network, it should provide for an alterable path control technique so that traffic can bypass damaged parts of the network without requiring the reinitialization of the network. This means that addresses on data streams (routing indicators) will be read at several places in the network and decisions made as to the path to be taken. This will allow EFTS to achieve the efficiencies associated with line load balancing but complicates path control with the problems inherent in a spill-forward system of routing, namely: shuttling or "ring-around-the-rosey." This problem is not severe in the single regional system described by the Federal Home Loan Bank Board, but is a potential source of trouble if the fundamental network architecture is not sound.
3. A third consideration in the design of the access network is *link control.* Link control is the acknowledgment of the receipt of data and the control of errors on each hop through the network. The two common types of link control are forward error correction (FEC) and Automatic Request for Retransmission (ARQ). In FEC, sufficient redundancy is added to the data stream to provide an acceptable degree of assurance that errors will be corrected at the receiving end without acknowledgment. The ARQ approach concentrates on detection of errors and the retransmission of a block of data when an error is detected. Synchronous Data Link Control (SDLC), a form of ARQ, is employed in devices conforming to IBM's Systems Network Architecture. A similar line discipline is emerging as an international standard and is likely to influence the design of future EFTS links. In an EFTS system, SDLC could be used between SNA devices, e.g., between an IBM 3600 Financial System and a thrift institution servicer's machine. Because of the variety of machines employed by the thrift institution servicers, however, switching systems will probably employ the line disciplines prescribed by the manufacturers for links between the thrift institutions and their servicers.
4. *Device Control* is the fourth element in the design of the network. Standards are required to control the operation of the terminal devices (paper feeding, spacing, indicator control, etc.). More important, however, are standards that determine the operation of intermediate devices, such as the transmission control units of the switch and the thrift institution servicers and the cluster controllers of the S & L's. These controls are necessary for network security and to pace data through the network to assure that the available data buffer capacities are not exceeded by bursts of data.
5. Finally, *end user protocols* must be developed for the EFTS. These protocols, used by the thrift institutions, are essential for customer security and include authentications and special codes agreed upon by the industry. As EFTS evolves, a parallel evolution of end user protocols can also be expected to evolve to support new services offered by the thrift industry.

Figure 2 summarizes the key controls in the network access subsystem by illustrating the format of an en route message nested within the control protocols described above. While every attempt should be made to reduce the communications overhead incurred by these protocols, the EFTS users will probably find the cost of this overhead to be low in comparison with the costs associated with loss of control of the network. Other controls such as modem synchronization and signaling through a dial-up network are considered to be within the responsibilities of the supplier of the communications link, while still other controls, such as end-to-end acknowledgment and password protection

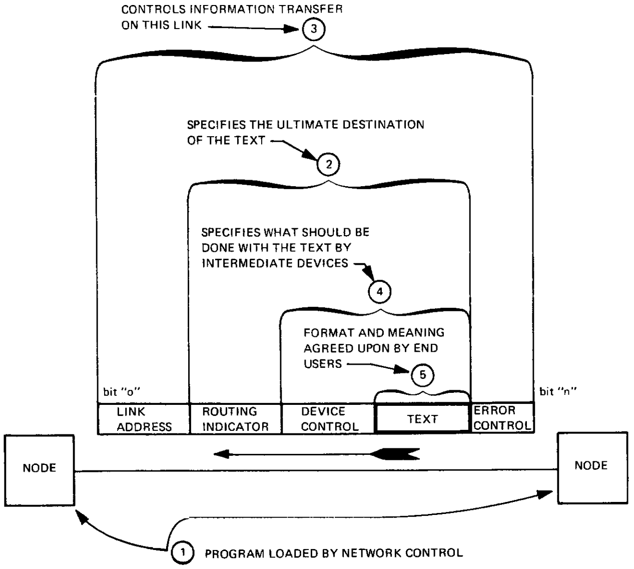

CONTROLS INFORMATION TRANSFER ON THIS LINK
3
SPECIFIES THE ULTIMATE DESTINATION OF THE TEXT
2
SPECIFIES WHAT SHOULD BE DONE WITH THE TEXT BY INTERMEDIATE DEVICES
4
FORMAT AND MEANING AGREED UPON BY END USERS
5
bit "o"
bit "n"
LINK ADDRESS
ROUTING INDICATOR
DEVICE CONTROL
TEXT
ERROR CONTROL
NODE
NODE
1
PROGRAM LOADED BY NETWORK CONTROL

Figure 2—Key controls on an en route transaction

are considered to be the responsibility of the thrift institution servicers themselves. These controls, while important, are outside the scope of this discussion.

Figure 3 illustrates the hierarchy of facilities that is envisioned for the Electronic Funds Transfer System. At the Switch and at the Thrift Institution Servicers, computer installations will be required to perform the switching and communications line handling functions. (The switching function can include circuit, message, or packet switching as well as certain hybrid combination of these techniques sometimes referred to as "virtual channel switching.")

At the S & L's themselves, the ADP equipment is visualized as cluster control equipment and terminals. Cluster controllers control terminal equipment which is either locally or remotely attached. Cluster controllers contain a pool of equipment normally associated with "intelligent terminals" and perform a data concentration function so that multiple terminals can share a communication line.

Figure 4 shows some communications options to be used in the design of the EFTS. It is reasonable to expect that much of the access network will be provided by various communications carriers under contract to the EFTS operator.

The availability of the services shown in Figure 4 and their tariffs are constantly changing so that no useful purpose would be served by repeating them in this paper. There are, however, several trends in communications that will affect the implementation of the Electronic Funds Transfer System, particularly the access subsystem.

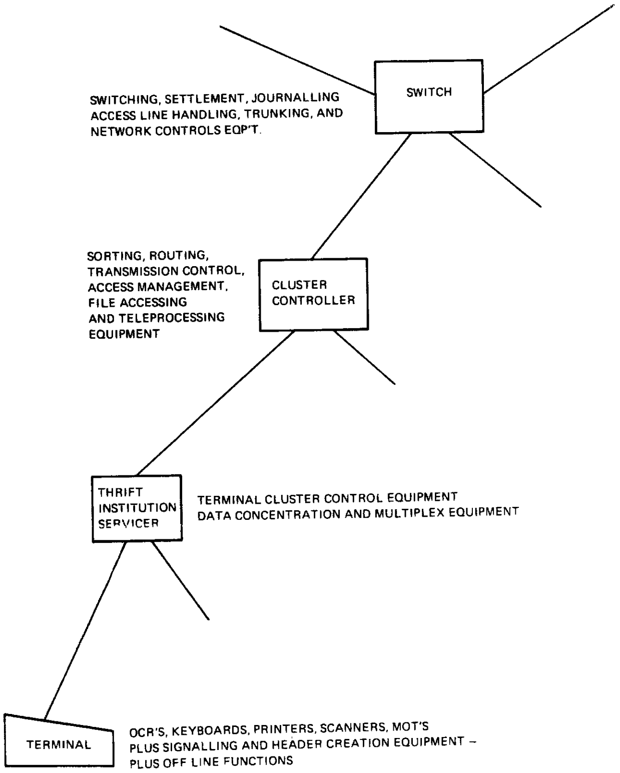

SWITCHING, SETTLEMENT, JOURNALLING ACCESS LINE HANDLING, TRUNKING, AND NETWORK CONTROLS EQP'T.
SWITCH
SORTING, ROUTING, TRANSMISSION CONTROL, ACCESS MANAGEMENT, FILE ACCESSING AND TELEPROCESSING EQUIPMENT
CLUSTER CONTROLLER
THRIFT INSTITUTION SERVICER
TERMINAL CLUSTER CONTROL EQUIPMENT DATA CONCENTRATION AND MULTIPLEX EQUIPMENT
TERMINAL
OCR'S, KEYBOARDS, PRINTERS, SCANNERS, MOT'S PLUS SIGNALLING AND HEADER CREATION EQUIPMENT – PLUS OFF LINE FUNCTIONS

Figure 3—ADP equipment hierarchy

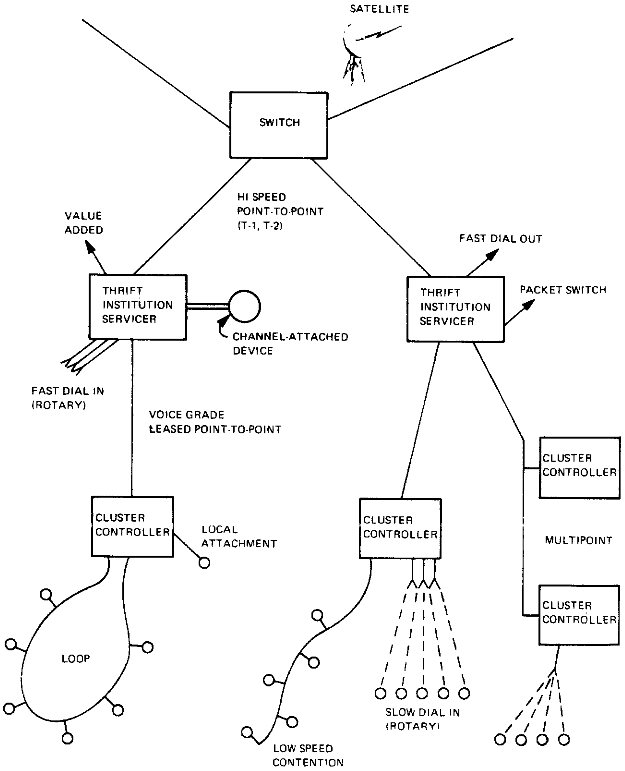


Figure 4—Some communications options

The move to digital communications

The communications industry is dominated by voice users who have traditionally been provided with switched analog circuits of approximately 3000 hz bandwidth. Digital users had to make do with the available services and adapt them to digital use through modems. Both switched and leased services are available under a variety of tariff structures. Data speeds range from the sub-voice level (50, 75, 100, 300 bits per second) through switched voice level 1200, 2400, and sometimes 4800 bits per second, up to 9600 bits per second that can be provided by present day modems operating over a leased, specially-conditioned voice-grade line. Higher speeds are available by leasing lines that have been derived from groups of voice channels (6, 12, 60 and 240 voice channels). New construction in the telephone system is employing digital communications techniques to an ever increasing extent. This is being done because the advantages of digital voice communications (circuitry is cheaper and easier to adjust) outweigh the disadvantages (up to 64 kilobits per second are used to provide high quality voice). The move to digital voice has an important effect on the design of the EFTS network. In the next few years both switched and leased digital service, at voice channel prices, can be expected to be available between major cities in the United States and will have channel capacities of 56 kilobits per second (derived from a 64 Kb/sec voice channel).

High speed access (T1) lines will also become available for data use at speeds up to 1.544 megabits per second and groups of these lines will probably be tariffed eventually to provide service up to the 96 megabit rate envisioned for digital television.

The use of satellites

Leased satellite capacity is an obvious candidate for the long haul trunking portion of the National EFTS of the future but a satellite system has a unique feature that may be advantageous even at the regional level. This feature is derived from an earth station's ability to "hear" everything repeated by the satellite transponder and thereby establish a fully interconnected or nodeless arrangement of earth stations.

This permits the design of a system that can set up and tear down point to point circuits on demand and can allow the EFTS to adapt to changing loads. Satellite capacity is currently available and several companies have planned satellite services that appear to be considerably more cost effective than equivalent ground service.

Distance sensitivity

For the past several years, the cost of providing communications service has become less sensitive to distance; that is, the cost of long haul communications has been falling and the cost of local terminations has been rising. These costs have not yet been fully reflected in the tariffs. The Government's encouraging of competition in public communications must inevitably result in tariffs that more closely reflect the costs. This will mean that the cost (to the EFTS operator) of a transaction will be relatively insensitive to distance and will be most influenced by considerations associated with the access subsystem (terminating arrangements, terminals, subscriber loops, etc.).

Dialup and switched networks

Until now, the use of switched communications for message service has been limited to low speed service over the teletype and voice networks. A trial offering of switched 50 kilobit service did not attract enough users and has been discontinued. As line speeds increase and become more widely available, dialup service may become an attractive method of communication in the electronic funds transfer system. Several new kinds of common carriers have been authorized within the United States and worldwide. These include specialized carriers offering a fast-dial fast-connect capability (under 1 second) and value-added carriers who lease lines from others and reoffer service by adding value such as error control, variable data rates, and switching schemes such as packet switching to attract data users who have highly intermittent traffic.

Special lines, loops, and channels

In addition to the conventional point-to-point communications lines available for the design of an Electronic Funds Transfer System, there exist several other kinds of access arrangements that warrant consideration. These include loops that reduce terminal polling loads by passing control from one station to another, high speed data loops being proposed for computer networks, and specially engineered customer-owned lines. Furthermore, there are no technical reasons that preclude the possibility of channel-attaching a thrift institution device directly to the switch. (Indeed, the concept of a "free standing" switch is technically unnecessary. If the member institutions were to agree, the switching function could be assumed by a thrift institution servicer as an addition to its regular functions.)

Complications to network control

As discussed previously, system control implies the ownership of terminals by cluster controllers, cluster controllers by higher level computers, etc. In a rigidly structured hierarchical tree network, the concept of ownership is simple. IBM's System Network Architecture, for example, provides for hierarchical system control in computer networks. If, as seems likely, the EFTS eventually departs from the tree structure, system control becomes more complex. Figure 5 illustrates a possible configuration of part of the EFTS network. If bank-to-bank links are permitted (or servicer-to-servicer), or if thrift institution servicers can access more than one switch, then the situation shown by Figure 5 occurs. Which switch is responsible for the integrity of the bank-to-bank communication? Does the thrift institution servicer authorize the dedication of a terminal to communication to a particular switch? Does it inform other servicers of the status of its subordinate terminals? These are the kinds of problems that must be resolved in the design of the access system control function.

Present-day computer networks usually call for some kind of control point, charged with the responsibility for configuration control, routing table distribution, statistics collecting, operator interface, system initialization, and other functions associated with the management of the network. The control point is sometimes responsible for controlling "sessions" between end users (terminals and computers). Sessions may be created at the time the system is generated (when the software is tailored to the needs of the network), they may be created at the time the system is initialized (when the network is first opened for traffic), or they may be created dynamically when the need for end-to-end communication is recognized. As the EFTS system evolves into a national network, dynamic session control from a single control point will probably be impractical. The EFTS can probably centralize most of the management at a single point (with appropriate backup) but the concept of sessions must eventually be modified to permit a terminal to be placed in permanent session with its "owner," a nearby, or regional control point designated when the system is initialized. This will permit distributed control and fixes responsibility for trouble diagnosis but, in turn, complicates the problem of routing transactions, guarding against lost transactions, and averting localized congestion caused by fluctuating traffic loads. These complications will be discussed next.

Complications to path control and routing

Figure 6 indicates some of the problems of network routing. If a thrift institution servicer has the exclusive responsibility for specifying the route that an EFTS transaction shall take through the network, it must be kept informed of the complete status of the network, including outages and congestion problems. The alternative is for the thrift institution servicer to

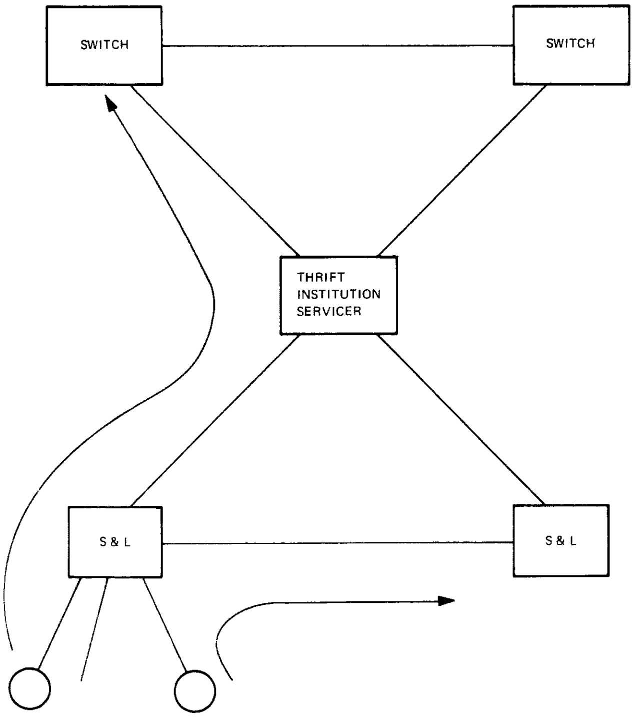


Figure 5—Complications to network control

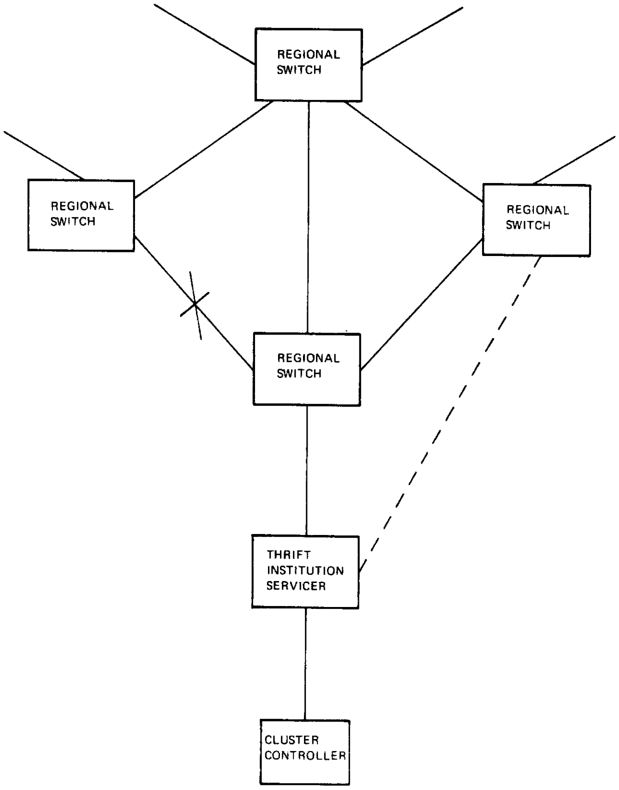


Figure 6—Complications to path control and routing

affix a routing indicator on the transaction and for the switches to assume responsibility for the path to be taken. If, as shown in Figure 6, the thrift institution servicer is permitted to home on more than one switch (This possibility is suggested in the Middletown scenario.) then the servicer must have the capability of determining which physical connection should be used to transmit a particular transaction. This logical decision may be influenced by time of day, information on the rest of the network, traffic loads and priorities.

If alternate routing is permitted in the National EFTS, then special precautions must be taken to prevent a transaction from circulating endlessly in the system. The technique currently employed in the U.S. telephone system involves a hierarchical system with alternate routing whereby a direct route is tried first, followed by an escalation to higher level of offices until a route is found. Lower-level trunks are engineered for high usage (i.e., the trunks are often busy and therefore efficient). This causes traffic to percolate up to higher level centers which are more generously engineered so they can cope with alternate-routed traffic that is generated by the overloads. Traffic engineering techniques developed over the years for the world's telephone system can be used to advantage in the design of the EFTS.

Complications to link control

Link control involves the transfer of a bit stream between two points. Problems in link control are dominated by the error rate of the communication line. If the line were perfect, or if the natural error rate of the line were tolerable because of some gross redundancy in the data being transmitted (as in a facsimile system) then link control would merely involve some gross protection against lost messages. For digital data streams with low redundancy, such as EFTS transaction, some form of error control is required so that a transaction can be accurately reconstructed at the end of each hop through the network. This can be performed by adding redundancy bits that are checked at the receiver for each transaction. Either variable length transactions or fixed length transactions can be used. The useful line speed of a link is a function of the raw speed and error rate of the line, the length of the transaction, and the amount of redundancy in the transaction.

Figure 7 shows experience of leased telephone lines typical of a few years ago. The system shown involves adding 8 error detection characters per transaction and retransmitting it if it is found to be in error. Notice that raw error rates are high for high bit speeds and that this requires extensive retransmission if the transactions are long. On the other hand, short transactions are also inefficient because of the overhead represented by the error control characters. In EFTS, because of the sensitivity of the data being carried, some form of error control will be needed. Because the line efficiency is sensitive to the transaction length used, careful attention to the error control problem is required before standards are adopted to minimize the effect of variations in the error performance of the communications link.

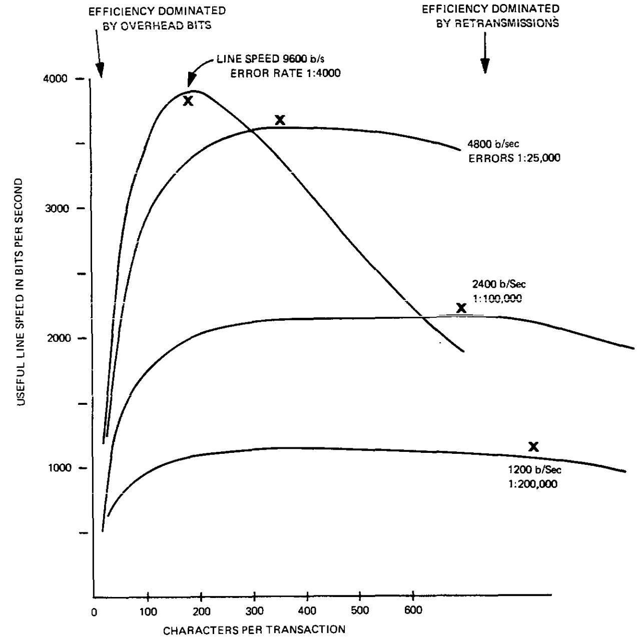


Figure 7—Complications to link control

Complications to device control

The problems of congestion in the EFTS will become severe when it evolves into a national system. In a data communications system with distributed control, overloads, when they occur, must be coped with by buffer storage capable of temporarily holding the traffic as it is moved through the system. Some means must be provided to prevent traffic from exceeding the output speed of the receiving device and from exceeding the buffer capacity of the intermediate devices (Refer to Figure 8). IBM's Systems Network Architecture accomplishes this through standardized messages among the intermediate devices that serve to regulate traffic through the system. Some similar set of device control protocols should be designed into even the early pilot models of the EFTS that will reflect the overload status of a device back through the network to control the flow of traffic through earlier stages.

Complications in end user protocols

The EFTS should be transparent to the users (i.e., thrift institution servicers). That is, once the addressing and input rules have been followed, the text of the transaction should have no effect on the operation of

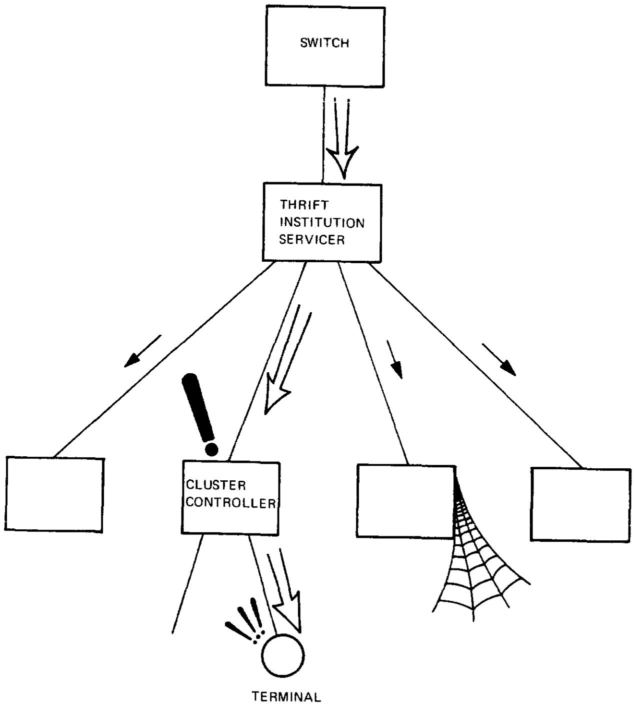


Figure 8—Complications to device control

the system, nor should the system have any effect on the transaction.

Such transparency is not easily achieved. Figure 9 illustrates four common pitfalls that cause problems. First, the link hardware may be affected by the content of the message. Consecutive streams of "1" bits or "0" bits sometimes cause problems. When these problems are overcome, the remaining problems are sometimes bizarre. There are instances of 27 asterisks causing a particular modem model to lose synchronization. In addition to link problems, the transmission control hardware may not be transparent. Certain dial telephone systems utilize specific frequencies in the transmission band for signaling and routing. These frequencies, if duplicated by the data stream, can cause misrouting. Routing software in intermediate devices can exhibit bugs years after its installation when some procedural change causes text to be interpreted as signaling information. Finally, there are incompatibility problems that may occur in the receiving devices themselves. The EFTS system is a long-term endeavor that will be characterized by the introduction of new devices over the years. It is essential at the start that a fundamental systems architecture be developed that will permit this evolution.

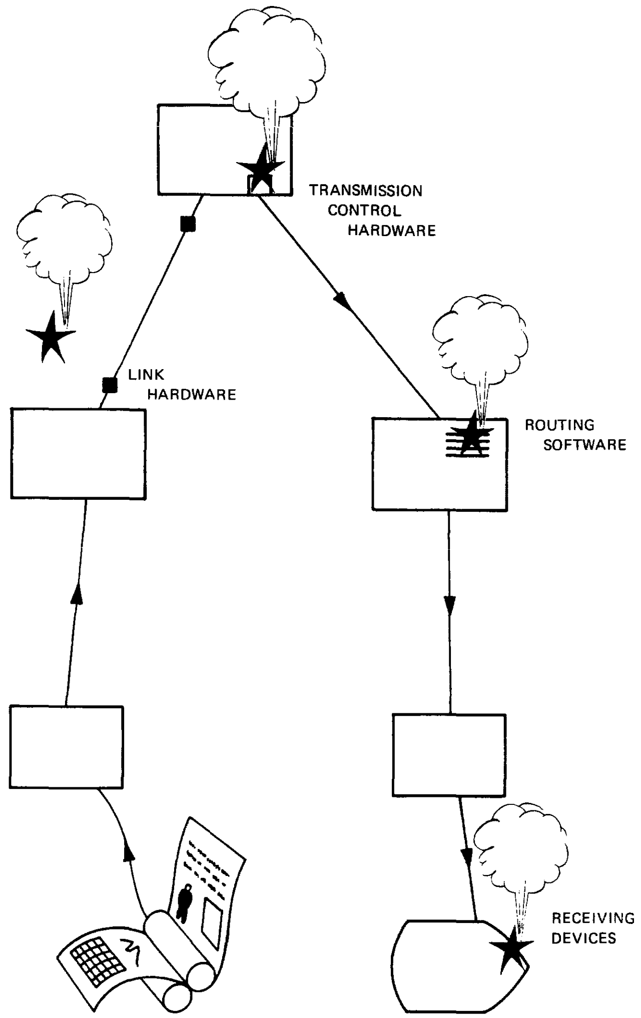
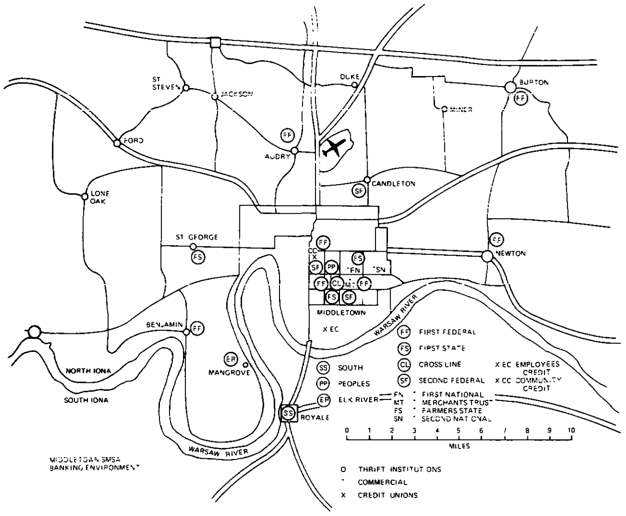


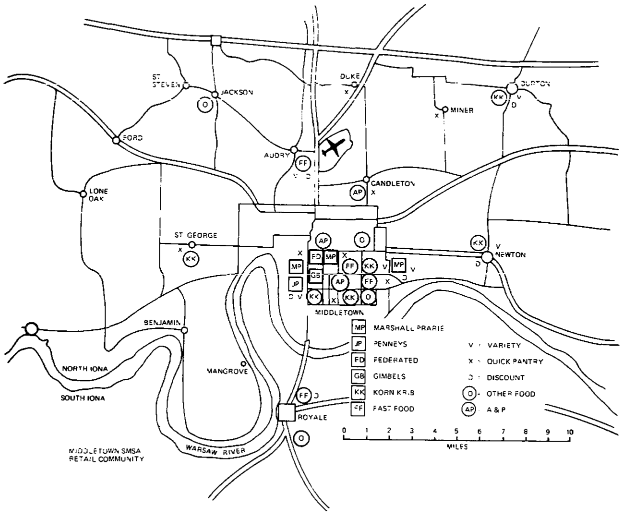
Figure 9—Complications in end user protocols

ARE COMPUTERS READY?

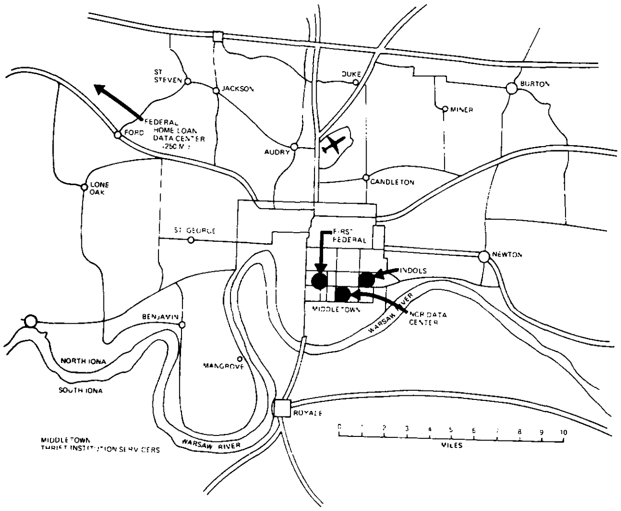
None of the problems discussed in this paper is insurmountable. Each data processing company offers a solution to at least some subset of these problems, and rival computer networking schemes have generated some lively discussions (some of which border on medieval theology). The Public, and the Banking, Communications and Computer Industries seem to be ready for the checkless society; yet the crucial component: A nationwide electronic funds transfer system seems to be an elusive goal. Comprehensive, cross-industry standards seem almost too much to expect at this time; but surely the computer industry can do better than it has in proposing at least the architecture for a bold, new approach to electronic funds transfer.



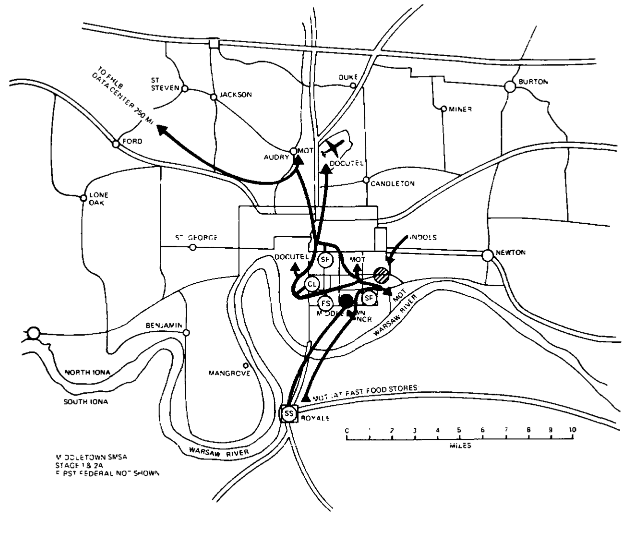
Attachment A



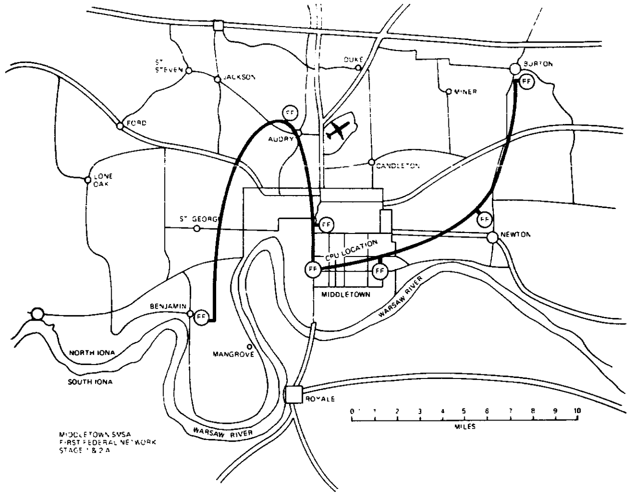
Attachment C



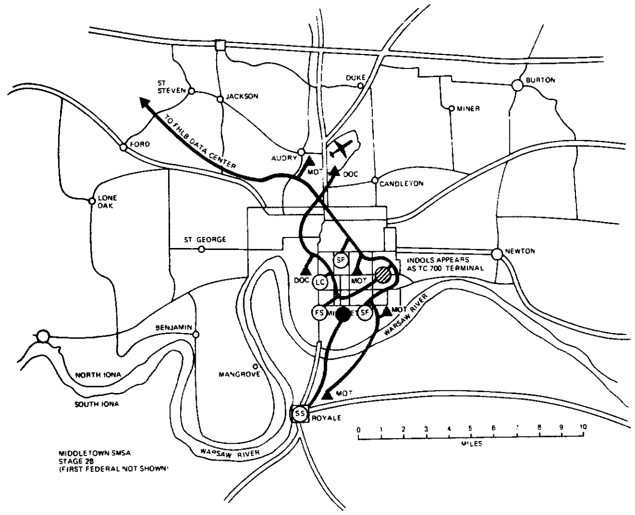
Attachment B



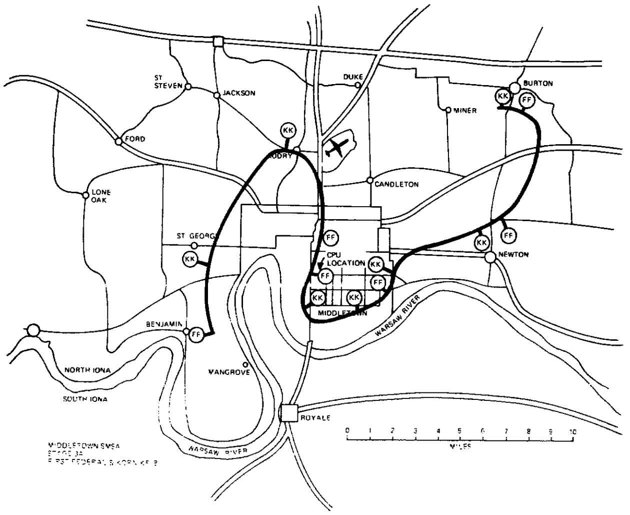
Attachment D



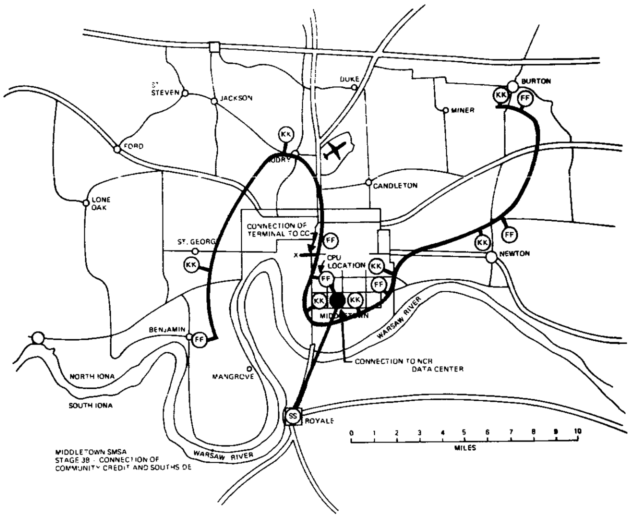
Attachment E



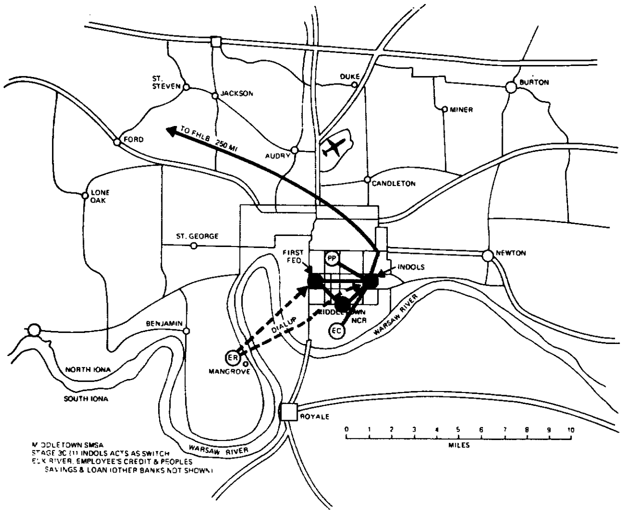
Attachment F



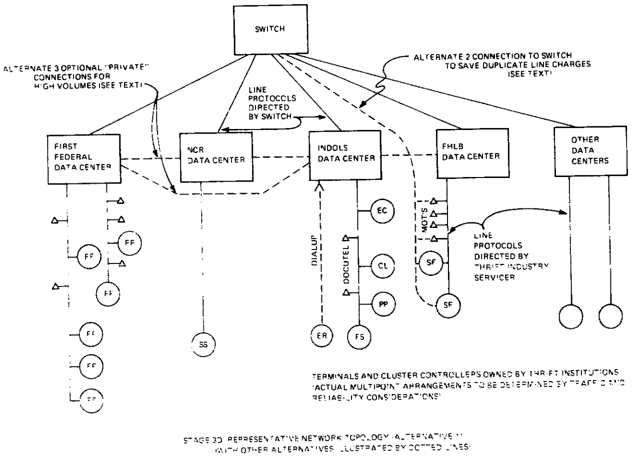
Attachment G



Attachment H



Attachment I


SWITCH
ALTERNATE 3 OPTIONAL "PRIVATE" CONNECTIONS FOR HIGH VOLUMES (SEE TEXT)
LINE PROTOCOLS DIRECTED BY SWITCH
ALTERNATE 2 CONNECTION TO SWITCH TO SAVE DUPLICATE LINE CHARGES (SEE TEXT)
FIRST FEDERAL DATA CENTER
NCR DATA CENTER
INDOLS DATA CENTER
FHLB DATA CENTER
OTHER DATA CENTERS
FF
FF
FF
FF
FF
FF
SS
DIALUP
DOCUTEL
EC
CL
PP
ER
FS
MOT'S
SF
SF
LINE PROTOCOLS DIRECTED BY THRIFT INDUSTRY SERVICER
TERMINALS AND CLUSTER CONTROLLERS OWNED BY THRIFT INSTITUTIONS
(ACTUAL MULTIPOINT ARRANGEMENTS TO BE DETERMINED BY TRAFFIC AND RELIABILITY CONSIDERATIONS)
STAGE 3D REPRESENTATIVE NETWORK TOPOLOGY (ALTERNATIVE 1)
WITH OTHER ALTERNATIVES ILLUSTRATED BY DOTTED LINES)

Attachment J

Personware

by H. W. BOMZER
University of Illinois
Urbana Champaign, Illinois

ABSTRACT

Personware is the human resource that makes it possible for hardware and software to perform the data processing function. Consequently, management is urged to invest in its maintenance and upgrading. The paper develops the effects of management objectives, user interface, organizational structure, job descriptions, and individual goals on a training program. Methods for implementing a program which will result in a high correlation of the skills inventory available, identification of skills required to meet functional needs, and specified employee objectives are discussed. Various educational techniques are shown to fit into a schedule which provides maintenance and upgrading of personware at a relatively low cost.

In the dynamic world of data processing management recognizes that successful performance involves not only hardware and software, but personware as well. In carrying out its plans for accomplishing increased productivity and greater efficiency, management cannot ignore the importance of the employees who perform the functions of the data processing business. Management seeks to satisfy the needs of its customers. Equally it must seek to satisfy the needs of the employees. It is only in this way that the organization can realize a full potential.

In this constantly changing world of data processing it is necessary for employees to be able to plan their futures. These plans should be consistent with the market demand for skills and the individual's ability to satisfy these demands. Inherent in his growth process is a recognition of one's own abilities, learning potential, behavior patterns, and needs as well as the anticipation of the employer requirements. With this in mind it is prudent to examine the present structure and needs for providing a frame of reference in which to plan for growth.

In providing for the professional development of its employees management is contributing toward the realization of its organizational objectives. This will occur if the career path plan which provides for training and progression is in consonance with the services of the organization.

During the past several years management has concentrated on acquiring larger, faster and more efficient machines. There has been increased emphasis on communications and on-line systems; higher level languages and special packages for retrieval have come into vogue; there has been greater sophistication in system development and the use of data bases. Concomitant with these changes have been reorganizations and realignment of personnel and their duties in order to effect efficient interaction and productivity. In this environment the data processing employee has become restless, sometimes frustrated and increasingly aware of the need for maintaining his technical skills. Not only is it necessary for the employee to maintain an adequate skill level, but he also requires a frame of reference in which to plan for growth. The objective of a career path is to provide the employee with the wherewithal to realize his potential and desires to the fullest while at the same time serving the needs of management.

How does management develop personware? This will vary from company to company, but in all cases it starts with the person and how he will fit into the organization. Management philosophy, its policies, procedures and structure are of considerable importance. For example, if the shop has programming teams as a modus operandi, then an individual who likes to "go it alone" may not function effectively in the team environment. A happy marriage for both management and the employee is predicated on mutual support.

Figure 1 illustrates schema that can be adopted to develop personware. It starts with management objectives. In terms of data processing, these objectives must be measurable.

Frequently, objectives set by management for data processing are beyond the control at the D. P. unit. Thus, management may state as an objective "Reduce the cost of inventory by X% over the next two years." Translated into tactics, this may involve the development of a new inventory system which substitutes on line input of data for an existing hand entry system. At the same time, management may set another objective "Do not increase the cost of internal service

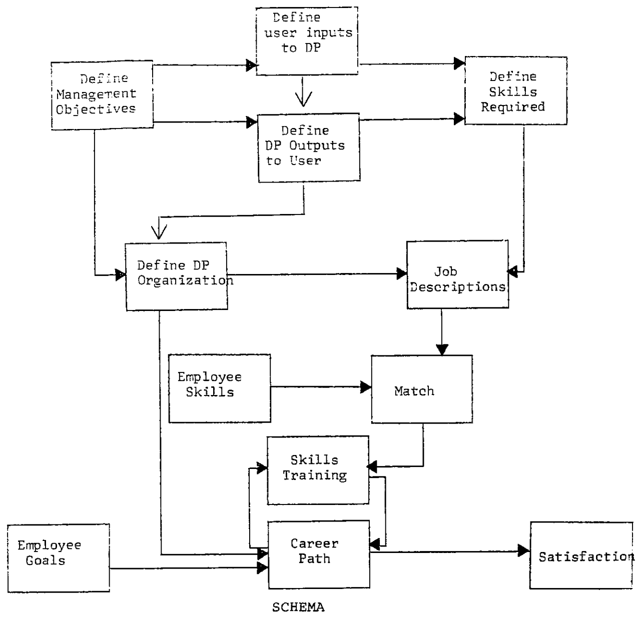


Figure 1

for the next year." If the data processing function in this organization is counted as service, management may impose a dilemma. The situation created by trying to satisfy both objectives can lead to frustrations which are not conducive to development of personware.

Within the D. P. organization objectives should be clear. To a large extent these are influenced by the D. P. budget. However, D. P. management should provide both long range and short range objectives of which employees are aware. Table I is a typical example.

A natural adjunct to objectives is a definition of services. The key to success lies with accepting input from the customer and producing output which meets his needs in a cost effective manner. The form of inputs and outputs affect performance as well as efficiency. However, from the employee point of view, they affect the skills that will be required to perform on the job. For example, consider a shop which operates strictly in a batch mode. Punched cards are the input and reports are the output. In this shop the analyst or programmer need not become proficient in terminal applications and languages. On the other hand, if it were a terminal oriented IBM shop, the analysts and programmers would be very much involved with such things as CICS, DMS, etc.

TABLE I

LONG RANGE

1. Realize in excess of 10% R. O. I.
2. Operate with centralized system and distributed network
3. Operate in Data Base environment within five years.
4. Provide security for user data

SHORT RANGE

1., Cut paper costs by 10% this year
2. Develop terminal oriented system for customer
3. Educate 2 analysts in Financial area to use Data Base
4 Identify data requiring limited access and install passwords.

When one considers the multiplicity of input and output techniques that are available, it becomes clear that training and education are continuing phenomena. The skills required for providing services and making transitions to improved and more efficient services must be inventoried. Training to maintain and upgrade employee skills are part of management's planning function.

A clear definition of the data processing organization is essential for the long term employee who envisions progress along promotional lines. Unfortunately, this is replete with traps and problems tied to wage scales and responsibility. Thus, a top notch analyst may aspire to a management position. However, there is no guarantee that even with proper training the analyst would be transformed to a good manager. Nevertheless, it behooves management to adopt an organization which can respond to its needs.

A typical organization chart is shown in Figure 2A. In this type of structure, a programmer entering the Applications section can see growth within the programming field. In fact, experience suggests that the hierarchy would require transferring from the Applications Programming section to the Analysis & Design section in order to progress to the manager position. However, the systems programmer would proceed by an entirely different route.

In Figure 2B the systems section is shown organized in a team approach. The lines of progression favor a project orientation. This construct produces

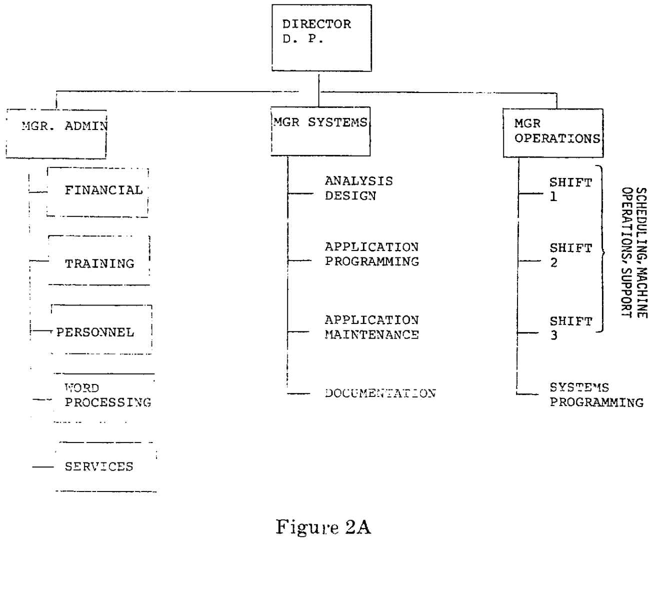


Figure 2A

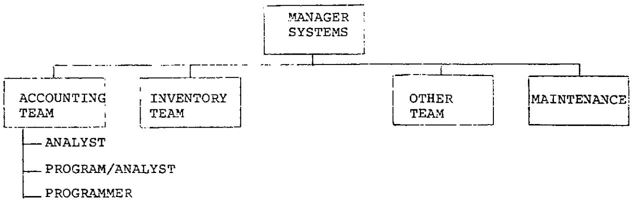


Figure 2B

people who are likely to be specialists in certain application areas. An employee who wants exposure to many application systems may want to carefully consider the policies regarding transfer from one team to another.

Personware, unlike software and hardware, is a self-evaluating mechanism. One of the references for performing the evaluation is a description of the incumbent's job. A "Job Description" would show where the position fits in the organization. It summarizes the responsibility and authority of each position, the tasks to be accomplished, skills and education required, and general information useful for evaluating performance. Figure 3 illustrates a job description format similar to that used as a guide for some Civil Service positions in Illinois.

A match of employee skills with job requisites provides a basis for implementing a training program. Depending on the size of the D. P. department, the amount of times scheduled for training and availability of employees for training, management may choose to offer several methods for accomplishing the desired results. Short courses are provided by several private firms who have developed instructional material. Lectures can be attended when the vendor schedules them at specified education centers; or, for large D. P. organizations, they can be scheduled on site. In addition to lectures, much of the basic material is available in self-study courses. Programmed instruction manuals represent one such format. Lessons are designed to instruct, quiz, and review as students progress through the text. In some cases, text material is augmented with practice "work type" examples. Another approach to training relies on multi-media. This technique usually involves a combination of reading, listening to audio tapes, viewing video tapes, and self tests in a guide book. To employ this technique requires an investment of $1500-$2000 in the video and audio equipment. However, it provides an extremely flexible means to provide training at various levels.

Management should be concerned with the progress of employees. Records can be kept by the personnel office, a responsible training office, or the employee's immediate supervisor. In a large department, a format for maintaining the records can be developed. The immediate supervisor should have the record available and use it to plan and encourage additional training.

Figure 4 illustrates a simple format which has been employed at the University of Illinois to maintain

JOB CATEGORY - CLASS, FAMILY

FUNCTION

ORGANIZATIONAL RELATIONSHIPS

DUTIES AND RESPONSIBILITIES:

1. KNOWLEDGE REQUIRED FOR THE JOB (State level.)
   A. EDUCATION
   B. EXPERIENCE
   C. SKILLS
2. RESPONSIBILITY
   A. SUPERVISORY CONTROLS
   B. GUIDELINES
3. DIFFICULTY
   A. COMPLEXITY
   B. SCOPE AND EFFECT
4. PERSONAL RELATIONSHIPS
   A. PERSONAL CONTACTS
   B. PURPOSE
5. ENVIRONMENTAL DEMANDS
   A. PHYSICAL REQUIREMENTS
   B. WORK ENVIRONMENT

COMMENTS:

Prepared by ______ Date ______ Reviewed by (Supervisor) ______ ______ AUTHORIZED SIGNATURE

Figure 3

SKILLS INVENTORY

ANALYSTS	ARCHI-TECTURE	PROG. FUND	JCL	FILE ORG.	CICS
Ray	3	2	2	3	1
Jim	2	2	2	1	1
Gary	3	3	2	3	1
⋮					
Jean	3	3	3	2	1

Proficiency Level: 1-Familial, no working knowledge; 2-Good working knowledge; 3-Proficient

TRAINING STATUS

Ray	C12-72	E1-76	-		-
Jim	C8-75	E1-76	-	S6-76	-
Gary	S4-76	C1-76	-		-
⋮					
Jean	C6-74	C1-75	-		-

ENTRY; E-enrolled; C-completed; S-scheduled; xx-xx is month and day completed.

Figure 4

training records. On a single sheet, the supervisor can see the inventory of skills required in the department, the level at which those skills exist, the training that has been taken to achieve a given skill level, and the status of training for each person in the group. Based on this chart, a schedule of training can be arrived at between the employee and the supervisor. Plans to reinforce skills or acquire new ones can proceed in harmony with production needs.

The accumulation of skills and worthwhile experience helps the individual develop. Management gauges the employee progress and provides the nourishment. However not all employees have the same goals, nor do all employees require the same nourishment. The career path plan should contain the characteristics for meeting the needs of both management and the employee. There are a number of ways by which the individual measures his goal attainment. These include status, responsibility, authority, salary, working environment, technical support, opportunity for improvement, and other fringe benefits. A career path plan provides the employee with a tangible reference for measuring his progress toward achieving his goals. The plan is somewhat constrained by the data processing organization and its philosophy. However the demands of the industry supply management with sufficient opportunity to reward the deserving employee. As the individual develops he becomes a valuable asset to the employer. The development of this "Personware" is potentially the highest payoff item in an operating budget. It deserves the attention worthy of this potential.

From data entry supervisor to data entry specialist

by CAROLYN M. DUNNING
Pertec
Santa Ana, California

ABSTRACT

For many years data entry lagged behind all other aspects of data processing. Originally, there were card punches where data was entered a character at a time. Then came key tape making buffering a common concept to data entry. Buffered key punches provided the same advantages of key tape, but had more flexibility. Then key-disk entered the scene and data entry became computerized. Data could be entered faster and more accurately than ever before.

What does this mean to the data entry industry? Operators do not have to be as well trained as before. But the greatest impact is felt at the supervisory level. 'Mini-analyst' training should be provided for those individuals interested in supervising key-disk installations. These classes should cover terminology, hardware components and some form of programming as well as the use of peripherals and options common to key-disk systems. Typical problems that can occur, both from an operator and the system should be discussed with methods to assist in analyzing them. Control procedures should be outlined.

Well-trained supervisors can help provide a more efficient data entry department and can be an asset in designing the data entry systems of the future.

For many years, the lowliest task associated with data processing was data entry. The manipulation and output of information progressed by leaps and bounds. Faster accounting machines, calculators and the advent of computers. Look how far computers have advanced in 20 years! The growth has been phenomenal. But what happened with data entry? Computers were processing more information at faster speeds every day. It was beginning to become a problem to find enough input to feed these hungry monsters. Why? Because data was still being keyed into cards one character at a time on equipment that was only as fast as the mechanics of the machines would permit. Keyboards had to be interlocked to prevent the operators from getting ahead of the card punches.

The introduction in the early 1960's of the IBM 029-059 card punches and verifiers produced two features that were indicative of things to come in the world of data entry. One was the automatic left zero option that gave birth to the concept of buffering data as it was keyed. Data could be keyed and corrected before it was actually punched into the card. Not only that, the operator didn't have to count and key leading zeroes. Granted, only integers could be buffered and the maximum was eight digits, but it was a beginning. The second feature that was new, was the photosensing on the 059 verifier. Cards could be read more quietly and much faster without waiting for any mechanical action to occur.

The next milestone for data entry was the introduction of the key-tape. Now an entire record could be buffered and massaged and corrected before being output. Just as important, the operator did not need to be inhibited by the mechanical limitations of the machine. Faster electronic keyboards were made which had a feature called n-key roll over. This 'remembered' keystrokes therefore assuring none were omitted, nor the operator locked out. Record sizes could be increased from 80 characters providing the ability to enter more data at one time.

Although key-tape machines were much faster, they had other serious limitations that made their use somewhat impractical for many applications. It was impossible for the operator to see an entire record all at once. Data would have to be read character-by-character. Another limitation was the inability to insert records within a batch. Missing records had to be appended to the end of the data batches which was not always feasible. Also, although both data entry and verification could be performed on the same machine, it was not convenient to do so. Since it is common practice for an operator not to enter and verify the same batch of data, this meant either switching tapes or machines. Therefore, key-tape was found to be very inflexible.

However, cards hadn't given up the battle yet. Then came the advent of the buffered card punches. Now an entire card image could be keyed into a buffer without physically punching a card until column 80 had been passed. While still being limited by the 80 columns in a card, the number of program levels was increased to provide the ability to key more data from a source document at one time. Now data entry had the best of two worlds; the speed and accuracy of key-tape with the flexibility of cards. Gone were the clumsy drums and their nemesis, starwheels. Programs could be stored in memory. Card punches could have the faster, electronic keyboards. As one card was being punched, the operator could begin keying the next one. Skips and dups were performed at electronic speeds. Operator statistics and the generation of batch totals was possible. Automatic emitting of data was available on some machines, as well as the automatic sequencing of program levels. Both punch and verifier were housed in the same machine. What a tremendous help this was in scheduling! No more worrying about the ratio of punches to verifiers. Throughput was increased tremendously.

Then all of a sudden, data entry grew up. Key-disk entered the scene and "Look Ma, we're computerized!" Now data entry had entered the same league as the big guys. Information is entered onto a disk with the assistance of a mini-computer. This data can be massaged and manipulated as much as desired. It is then usually written to tape for the mainframe computer to process. On most systems, it can optionally be transmitted directly to the mainframe, either over a telephone line or a direct interface. The advantages to this type of system are numerous. Data can be validated as it is keyed. Calculations can be performed and the results can be automatically entered into the records. Likewise, constant data. Now it isn't a question of keystrokes, but truly a matter of throughput. The data entry department has finally become a respected member of the data processing community. And this is just a beginning.

What does all of this mean? First of all, operators do not need to be as well trained as with the earlier key punches. Although key punch keyboards are standard, optional typewriter keyboards make it easy for typists or clerical personnel to become data entry operators. Information can be easily entered at the source, eliminating or greatly reducing the possibility of errors, which typically increases turn-around. Programming is no longer a factor since most systems are 'preprogrammed' with all operators using the same programs. Extensive procedures need not be remembered as most key-disk systems can prompt the operators as well as alert them to error conditions.

Well and good for the operators. But what happens at the supervisory level? Here is where the greatest impact is felt. Now to be really effective and utilize a key-disk system to its greatest potential, the supervisor should be a 'mini-analyst' with a fundamental knowledge of basic computer systems and data management. Gone are the cards with the visual evidence of work that has been completed. Data is now in a 'never-never land' called a disk. But how do we really know it is there? What does it look like and what happens to it now? These and many more questions plague a supervisor who is frequently answered with a "Don't worry about it. Just do exactly what you have been told to do and everything will be alright."

Key-disk has been over-simplified by many vendors. Who wants to frighten the supervisor with a lot of confusing terminology? After all, a very competent mainframe programmer wrote all of these programs and all that needs to be done is for the operators to enter and verify the data, the supervisor to write it to tape for the computer and then delete it from the disk. Simple, right? But what is wrong with this logic?

Let's start with programming. There's a world of difference between a program for a mainframe where an operator selects it and it executes with very little interference, and a program for a key-disk system where many operators sit and use it for hours at a time. A poorly written mainframe program can be slow and inefficient memory-wise, but this has little, if any, effect on the operator. On the other hand, a badly written key-disk program may not only be inefficient as far as the processing goes, but it can be very tedious for an operator to use. Who is best qualified to write a data entry program? A mainframe programmer or systems analyst whose primary key punch experience comes from punching their own programs (usually freeform) while in school or a key punch operator or supervisor who has been making program cards and punching and verifying data for years. These people know what is necesary for entering data easily and accurately and should be best qualified for writing programs that are going to be used by all operators on a system.

Another area of concern that is created by a key-disk system is the terminology that is used. A supervisor must be familiar, not only with standard data entry terms, but also those associated with computer operations. While most key-disk manufacturers have attempted to keep the terms data entry oriented and as simple as possible, this is not always feasible. For example, what about tape writes with blocking factors, translations, labels, etc.? And how about bits, bytes and words? We didn't have all of that with cards. The worst problem we had may have been trying to translate the holes in the cards into characters if the print mechanism failed. Otherwise, we just handed a deck of punched and verified cards to the computer operator and everything else was taken care of.

Some key-disk systems must compile programs before they can be used. That really is scary. With card punches all we had to do was create a program card, either put it on the drum and lower the starwheels or read it into memory and we were ready to go. If the program card wasn't quite right it was a simple matter to change it. Simplicitly was the key word.

What is the point of all of this? Well, many schools and colleges provide very extensive computer training, both for programming and operations. And some schools also provide key punch training. But what about a person who is interested in learning about a key-disk system? Oh yes, manufacturers provide training, but only on their equipment and usually only for customers. Normally, these are rather abbreviated and do not cover a lot of things a key-disk supervisor might want to know. A mainframe programming or systems class taught at the college level is more than most data entry supervisors would need, or want. They aren't interested in becoming mainframe programmers. So why not provide educational classes tailored to suit the needs of a key-disk supervisor? Data entry is advancing at a rapid pace and experienced, qualified individuals are necessary to create efficient key-disk departments. Again, who is more qualified to handle this task than an operator or supervisor who has been entering data for a long time and understands the needs and problems associated with data entry? This is a 'people-oriented' environment more so than any other area of data processing. It can be a very emotional situation.

What kind of course would this be? It does not need to be concerned with all of the problems of a large mainframe computer, but should be more like a 'mini-analysts' course, tailored for individuals who want to work with some type of computerized data entry system. It must be generalized to cover the common features and functions of most key-disk systems, which is a large order since none are totally alike. However, there are many characteristics that are common to all, or most, systems.

An explanation of computer concepts and terminology is vital. Bits, bytes, words, binary, ASCII, EBCDIC, software, hardware. These are terms that are thrown around by the customer engineers or systems analysts working with a key-disk system. But what do all of these words mean to the supervisor? I think an understanding of some of these terms can help the supervisor communicate better with the people who have to answer questions and service the system. It will also help allay some fears of the 'unknown'.

The difference between hardware and software should be explained. As any one who has been in the real world of data processing knows, it is often very difficult to isolate problems between software and hardware. A supervisor who has some understanding of them can possibly better define a problem that may be occurring with the system. The hardware components that are part of the system should be explained, the most common factor being the disk. This should be covered in sufficient detail so the supervisor has a clearer understanding of its function. Tracks, sectors and AU's should be explained, as well as some explanation of a volume table of contents (VTOC). Some systems require the disk be initialized regularly. The purpose of this and its effect should be explained. Most systems also use some type of WARM and COLD START procedures. The difference between the two functions should be defined as well as what will happen if the wrong option is accidentally selected.

Tape drives are very important to most key-disk installations. Their care and use should be thoroughly covered. Proper cleaning and usage of both tape drives and tapes is very vital to producing good, clean output as the improper handling of tapes can create many errors from the data entry department. The cleaning of tape heads and a recommendation of what should be used to clean them is important. The benefit of having tapes certified regularly should be discussed. Since most drives are loaded and unloaded in much the same way, this could be demonstrated with students actually mounting and dismounting their own tapes. Terminology connected with tape drives should be discussed. For example, the difference between 7-track and 9-track tape drives should be explained, bits per inch and the difference between 800 bpi and 1600 bpi as well as parity and its purpose should be covered. An explanation of the appearance and purpose of BOT (beginning of tape) and EOT (end of tape) markers and how are they sensed is vital. Cutting the lead from a tape and putting on a new BOT marker should be demonstrated. EOF (end of file) should be defined. These are things that will confront a supervisor who may have no idea as to what is meant.

A most important aspect of any key-disk training is some form of programming. Although all systems use their own type of programming, there are many common factors that can be discussed in a generalized way. Most systems use at least some type of 'checkbox' programming. This is usually merely a fill in the blanks procedure to tell the system what is wanted. It usually contains field descriptors, the most common being field size and shift (alphabetic, numeric and in some systems, lower-case alphabetic). Other controls can also be defined in the checkboxes. One of these might be data type which specifies exactly what types of characters can be entered into a field and provides a character-by-character validation as the field is keyed. For example, a field may be programmed as alphabetic, but numerics and blanks are also allowed. Data type would have to specify this. Or perhaps a field can only be numeric, with no alphas or blanks permitted. Another common control is how the field can be exited. Perhaps it doesn't need to be exited at all and as soon as it is completed, control automatically goes to the next field. Or maybe a field release must be depressed before continuing to prevent overflowing into the next field. The extent to which a field must be keyed may also be specified. Perhaps a field must have every position keyed (none can be blank), or maybe it must have some data keyed, but not necessarily be filled. Maybe it must not be keyed at all and the system will insert some type of constant or calculated data. Justification and fill can be programmed so the operator does not need to remember whether a field must be right-justified and zero-filled or left-justified and blank-filled. The system will handle it automatically. Range checking and batch balancing frequently can be programmed in a checkbox program.

Since most key-disk systems allow rearranging the data for output (commonly called reformatting), fields can be organized so that it is easier for the data entry operator. Various types of verification can be programmed, such as key, sight, to-balance or not verified. In many instances, verification can be greatly reduced, or even eliminated.

In addition to checkbox programming, many systems use some type of higher level programming language, the most common being similar to COBOL. It might be well in a class of this type to cover the general concepts of computer programming, particularly the logic of COBOL such as IF, MOVE and the arithmetic statements ADD, SUBTRACT, MULTIPLY and DIVIDE.

Most key-disk systems have some method of prompting the operator. The purpose and general use of prompts should be discussed, such as the most effective way to program them so that they are meaningful to the operator. Peripheral equipment such as teletypes and line printers should be discussed. Since each system probably utilizes these options differently, their use can only be discussed in a general way. However, feed controls, carriage tapes, top-of-form, and other common characteristics might be explained. Since most data is usually entered in an unedited form, (no commas, periods, dollar signs, etc.) and printed reports will have these characters inserted, editing might be mentioned. Examples of the most common editing functions should be provided.

With the increasing use of data communications, a discussion of this feature might be appropriate. Since this is a very complex area it must be generalized and greatly simplified. A brief description of the most common protocols might be good as well as factors necessary for initiating a transmission or receive.

Since most systems use some type of full CRT display, the generalized areas of display should be discussed. For example, there are usually one or more status lines, a message line and several lines for displaying data. An explanation of what is displayed in these three areas is appropriate.

A discussion of error messages and the most common types of operator errors is necessary. This might include common errors such as keying an invalid character, attempting to key too many characters in a field or keying data that is not within a specific range. Most systems also provide the capability of creating some types of individual 'tailor-made' messages. A recommendation of what these messages could be and where they should be used would be helpful.

Relatively common system problems should be covered. This might include such things as not being able to bring up the system, a terminal or entire system hanging or not being able to locate a batch of data or a program. Things to look for when any of these conditions occur should be outlined. It can help a customer engineer isolate a problem if a supervisor has previously tried to analyze what is happening. Common operator problems should also be explained, such as keying an incorrect batch name or number, or opening a batch under the wrong mode. Most systems also have some type of search procedure. The parameters available should be discussed.

The supervisory functions are basically the same for all systems. The purpose of the most common ones should be explained. This might include things such as tape writes, deleting data from the disk, system saves, obtaining a disk status, assigning a new name or batch number to an existing batch of data, printing to a line printer or requesting batch status. Generalized operator statistics should be discussed.

Control procedures are a very important area of a key-disk system. Batches must be carefully and accurately logged and monitored as the work progresses. The log must be periodically checked against what is on disk and the status of the batch. (Has the batch been completely entered? If so, has it been verified? Does it contain any invalid data? Is it in- or out-of-balance?) These are things that are very important when the data is output. When a write is performed, batches must be checked against the log to be certain that all desired batches were written, and no extraneous batches were included. Disk status must be checked regularly to assure that the disk does not get full since in most instances, this can create severe problems. Deletes must be very carefully monitored and system saves regularly scheduled. System crashes where data is lost is not a common occurrence, but just one crash can be a disaster in that hours, or even days of work can be lost if the data is not properly backed up.

Well-trained key-disk supervisors can be a great asset to the industry in optimizing the quality of data going to mainframe computers. They can also be very instrumental in providing good, valuable input for creating better data entry systems in the future. Data entry is still in its childhood. There will be many changes in the years to come and the term 'data entry specialist' will be just as common to data processing as systems analyst or programmer. Now is the time to start training these specialists for tomorrow.

A modern beginning programming course*

by ROY F. KELLER
Iowa State University
Ames, Iowa

ABSTRACT

This paper describes a beginning programming course. It represents an approach to the "right way" to teach programming independent of any programming language. This is accomplished by thinking of programming as a two part process—(1) constructing an algorithm and (2) translating the algorithm into a program in some chosen programming language. Basic structured programming constructs are used for constructing an algorithm and translation is demonstrated by translation of control constructs into FORTRAN.

INTRODUCTION

The following quotations are a good way to introduce this paper:

(1) "Over the past decade, computer science has suffered a loss of innocence. No longer can a programmer be a gullible optimist, convinced of a program's perfection by success with a few chosen data. A program must be seen to be *correct*, and *clarity* has become essential. One of the keys to clarity is the set of control structures used, and the debate over the choice has been lively." Ledgard and Marcotty.[1]

(2) "One point should be made clear. We must distinguish between the programming language and the notation used while programming. One doesn't program *in* a language but *into* it." Gries.[2]

(3) "Yet it is fair to say that almost none of the elementary programming books say anything about problem solving, about orderly thinking, about expressing algorithms clearly and simply. The only conclusion to draw is that the students are not being taught how to program; they are being taught a language." Gries.[2]

(4) "It is insufficient to present the endproduct and expect the beholder to perceive its structure by inspection or even deep meditation. Instead, the beholder must also see at least part of the programmer's thought process, starting from the original (very abstract) version and proceeding to the end product via a clearly presented sequence of clear transformations and refinements." Denning.[3]

(5) "We have outlined three views of what a program should be. The final form of development is suitable for communication to a computer; a carefully documented sequence of forms is suitable for communication between programmers; a proven sequence of forms is suitable for communication to posterity." McKeeman.[4]

(6) "I now believe that it was a fundamental error to feel that we could think in a programming language." McKeeman.[4]

(7) "I would like to see the S.P. forum redirected towards a less hopeless task than finding the perfect programming language or formally defining S.P. itself. If we take the definition to be simply the construction of efficient, readable, understandable, modifiable, and verifiable programs, we can discuss ways to globally reach these goals by educating the people who do programming. Since it follows from the axiom that no amount of construct-clipping will make the typical graduate of a "data processing" school even a potentially good programmer, we must find something that will. What can be said to be the *proper* use of goto's, conditionals, and global variables (and block structures, pointers, etc.)? What are general guidelines to follow with respect to procedures? How does one go about modularizing a task?" Flon.[5]

The ideas in these quotes along with those expressed in the many papers dealing with programming and structured programming[7-12] form the basis for the following ideas in developing a beginning programming course:

a. The primary issue in programming is one of constructing a solution to a given problem to be programmed.

b. The solution should be "correct" and "readable." (Where correct means the solution solves the problem intended and readable means the form of the solution is "easily" understood by people.) Concepts of efficiency and reliability in programs come after correctness and readability. These ideas should come later in a more advanced course. However, a beginning course

* This work was supported by the U.S. Energy Research and Development Administration under contract no. W-7405-eng-82.

should promote practices which lead naturally to higher level considerations.

c. Constructing a correct and readable problem solution can best be done through a sequence of refinement steps starting with an abstract form and ending with a form easily translated into most any high level programming language.

d. In the refinement process problem solving constructs for constructing *well structured algorithms* are essential.

e. The purpose of a programming language is so that a problem solution can be expressed in a form understandable by a computer. A programming language is not often a good language in which to think or to document. It may be that expecting a programming language to serve both as a language (1) in which to think and (2) in which to make a computer understand is expecting too much.

A beginning programming course based on the above ideas has been taught since Fall 1974. This paper reports on the course contents and to some extent on its success as reflected by student and instructor surveys.

BASIC PHILOSOPHY FOR A BEGINNING PROGRAMMING COURSE

Based on the above, our approach to teaching programming was to focus carefully on what is essential to constructing a "well structured" solution to a problem in the form of an algorithm. What problem solving constructs are needed? Is there some hierarchy in such constructs so they can be developed naturally?

We divide programming into two parts:

Part I: constructing a "well structured" algorithm,
Part II: translating the algorithm into some programming language. (a program)

An algorithm is defined to be a problem solution, in the form of a sequence of statements which can be executed by a human using any devices at his disposal. A program is a problem solution in a form understandable by a computer. A "well structured" algorithm is to be a *correct* and *readable* one.

Dividing programming into the above two parts makes it possible to teach much of programming independent of any particular programming language. Part I is problem dependent and Part II is programming language dependent. In addition Part II can be taught by showing how to translate a particular Part I construct into appropriate statements in a chosen programming language. Thus the details of a language can be studied in the context of how to use it.

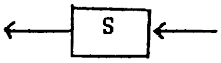
Our philosophy for a beginning course is to naturally develop problem solving constructs for constructing correct and readable algorithms and then show how to translate these into language statements in such a way to preserve the structure of the algorithm in the program. Thus we have "structured programming" in any programming language.

CONSTRUCTS FOR CONSTRUCTING ALGORITHMS

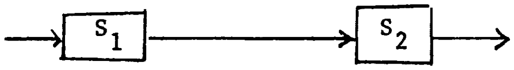
The details regarding the development of a set of constructs and pedagogically how to develop these can be found in Keller.[6] Here we shall list them along with some explanation or reasons for the construct. (All constructs are developed in context of a problem to be programmed starting with problems requiring simple constructs and progressing to the more complex ones.)

a. Basic constructs (or concepts): value, constant, variable, identifiers (names), expressions (operators and operands), statements, assignment statement, read and print statements. The crucial concepts here are the "ideas" of *a value, a constant, a variable* and of value and name association. Initializing values and assigning values form the basis for programming.

b. Decision making constructs (control mechanisms): an action (execution of a statement); sequence of actions, if-then and if-then-else constructs; while-do and repeat-until constructs. These constructs can be represented by flow diagrams and developed naturally in the following order.

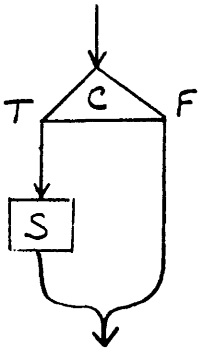
(1) action 

(2) sequence of actions

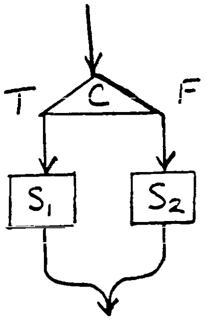


(3) if-then: *if* C *then* S

(Where C is a condition and S is a statement or sequence of statements.) The concept is to do S zero times if C is not satisfied (F) or once if C is satisfied (T).

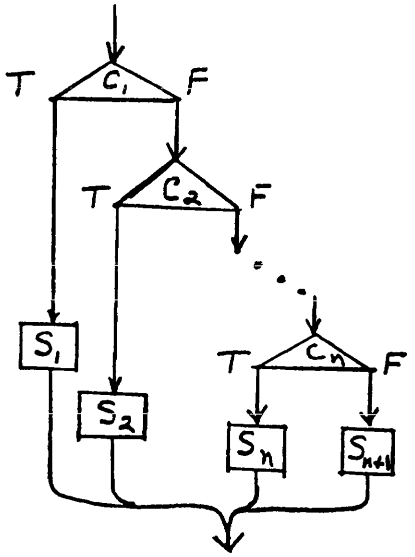


(4) if-then-else: *if* C *then* S_1 *else* S_2



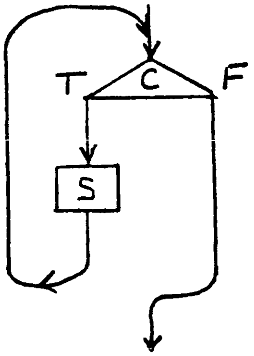
The concept is to do S_1 if C is satisfied; S_2 if C is not satisfied. Always one, S_1 or S_2, is to be done.

(5) nested if-then-else: *if* C_1 *then* S_1 *else if* C_2 *then* S_2 *else* . . . *else if* C_n *then* S_n *else* S_{n+1}.



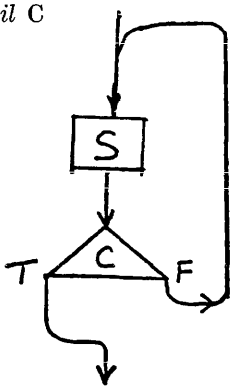
The concept here is to select one sequence to be done from many (more than two). This represents a natural extension of if-then-else.

(6) while-do: *while* C *do* S



The while-do concept develops naturally from if-then. If-then implies S is done zero or one time and the while-do extends doing S to more than once, i.e., iteration, while still preserving the idea of doing S zero or one time.

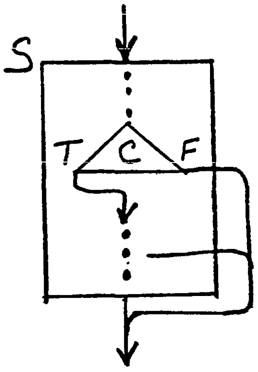
(7) repeat-until: *repeat* S *until* C



The concept is to do S repeatedly until C is satisfied.

These are the D and D′ structures (without the case construct) found in Ledgard and Marcotty[1] that have been studied by Dijkstra and others (See Reference 1 and references therein for further details on control structures.)

(8) repeat-forever: *repeat-forever* S



The concept here is to iterate forever through the sequence S which will contain one or more *exit* constructs of the form:

if C *then exit* or *exit* alone

Exit is always to the statement following S. In a beginning programming course there is seldom a need for more than two exits. This construct is a natural extension of while-do and repeat-until and obviously includes them.

c. Modularizing constructs: (Blocks and procedures). This includes:

(1) the concepts of local and global variables and how to specify their scope using some form of a declare statement;

(2) procedures as independent blocks with information passed through the formal and actual parameters. (This includes function procedures.)

The essential point for the student to understand is that modularizing comes naturally when one constructs an algorithm in a refinement fashion.

d. Data structures and value types.

Throughout all of the previous subject matter simple data structures (simple variables) are implicit. Iteration is developed by introducing a one-dimensional array data structure, introduced by considering a problem to be programmed. Multiple dimensional arrays and multiple dimensional structures are next developed. At this point the goal is that the student see clearly that developing an algorithm starts with his visualizing the data structures which are appropriate to the problem and naming these structures in such a way that he has names for substructures. He then must write statements involving these structures which when executed will solve the problem intended. The above constructs are his means of getting the job done through a sequence of refinement steps.

Character and Boolean values are next introduced again by considering appropriate problems. These data values give rise to taking a closer look at expressions, in the sense of operators and operands, and expression evaluation.

e. Verification of algorithms.

Throughout the refinement process of developing an algorithm, verification of each refinement step is heavily emphasized. Most of this is "hand" verification, testing decision points and iteration termination. In some cases simple proofs can be inserted to show a part of an algorithm (module) is correct.

Each refinement step is followed by the programmer proving the refinement is correct either formally or experimentally by hand execution with trial data. If great care in verification is taken, then a minimum number of bugs are introduced in an algorithm and programmer time is used efficiently.

Refinement steps and verification are never tidy. Many trial refinements are tried, determined to be incorrect by trial verification and replaced by other trial refinements. This is what programming is about; the student learns by doing. This is how he learns to *think* about the process of constructing a program.

TRANSLATING ALGORITHMS TO PROGRAMS

All the previous ideas are based on problem solving constructs needed for constructing algorithms which are correct and readable. These are essentially independent of any major programming language.

At this point any programming language could be introduced and studied from the standpoint of translating problem solving constructs into programming language statements. Of course it would be great if the major programming languages had these constructs. Unfortunately they do not. Someday major programming languages will contain these constructs, however, there will still remain some translation effort necessary for syntax reasons alone. In constructing an algorithm the programmer can ignore many details that a computer requires, but, a person does not. He can use indentation, one statement per line, etc. to make the algorithm readable for a person. Many of these human readable forms may never be machine readable. So some translation will be required for machine readability. The point is that *people read algorithms and computers read programs.*

The translation process is really not very difficult even for a language similar to FORTRAN. We illustrate translation to FORTRAN of the decision constructs using the FORTRAN logical IF statement: (These and other translations can be found in one of References 13-17.)

a. *if* C *then* S

```
        IF (C) GO TO then
               GO TO next
then  S
next  CONTINUE
```

Where *then* and *next* represent statement numbers and C must be translated to a logical expression in FORTRAN.

b. *if* C *then* S_1 *else* S_2

```
        IF (C) GO TO then
               GO TO else
then  S1
               GO TO next
else  S2
next  CONTINUE
```

(Where *then, else* and *next* are statement numbers.)

c. *while* C *do* S

```
while IF (C) GO TO do
             GO TO enddo
do S
             GO TO while
enddo CONTINUE
```

(Where *while, do* and *enddo* are the statement numbers.)

d. *repeat* S *until* C

```
repeat S
    IF (C) GO TO next
             GO TO repeat
next CONTINUE
```

(Where *repeat* and *next* are statement numbers.)

e. Repeat-forever translation is obvious. Exit translates to a GO TO *exit.*

Translations for variables, constants arithmetic expressions and Booleans expressions are straightforward. Procedures and functions procedures translate to subroutines and functions respectively in FORTRAN.

Blocks do not translate directly into FORTRAN; local variables must translate into global ones unless a subroutine or function is used for translation. Other programming languages avoid many of the FORTRAN translation problems.

While the above translations represent good ones they are by no means the only ones possible. These translations are the most natural and keep the structure of the algorithm present in the program. By use of comments in the FORTRAN program the translations can mirror even clearer the structure of the algorithm. Our experience has shown this is not of much use provided the final algorithm and selected refinements are maintained along with the program for documentation purposes; again the idea is that people read an algorithm and the computer reads a program.

SUMMARY AND CONCLUSIONS

This type of course in programming using FORTRAN has been taught to approximately two thousand students mostly non-computer science majors by large lectures (150 plus students) and small (25 students) recitations. Lectures met twice a week; recitation once a week. Fall 1974 was the first time.

General reaction from students has been favorable. For the first time in our experience students feel they are *learning* to program and feel they can. About sixty percent of the students reported in a survey that they felt "competent" to construct a program in FORTRAN. In a survey reported on by Krietzberg and Shneiderman,[18] in the preface of their book, only ten percent reported they felt "competent" to construct a program. It seems clear that the emphasis on constructing algorithms by refinement and verification is what makes the difference.

A total of five faculty members and more than twenty graduate assistants have been involved. With few excepts all agree that algorithm construction is most crucial and should be the emphasis in a beginning course. Most feel that regular size classes (about 30 students) would be much more effective than large lecture sections. It is very difficult to teach problem solving via large lectures.

Both students and staff feel that two languages are involved and this seems to create some difficulties for the student. The top students overcome this easily, but, others have troubles. It takes a while before the student realizes that the problem solving constructs are for thinking about the problem and its solution and the programming language is for translating into.

Further support for this type of programming course can be found in the conclusions of the paper by Ledgard and Marcotty.[1] We list their four points here. More detail can be found in their paper.

1. "From the programmer's viewpoint, theoretical results based on the conversion of one program form to another under restrictive conditions may not be practical significance."
2. "The need for higher level (above D and D′) control structures remains unproven."
3. "The utility of the goto is seriously questioned."
4. "The utility of D′ structures over D-structures is supported." (D-structures are: actions, compositions, if-then-else and while-do. D′ structures are: any D-structure, plus if-then, repeat-until and case statements.)

We feel all of these except the case statement are important for the beginning programming course. Case statements and other control structures (BJ_n, RE_n etc. See Reference 1 for definition of these and other control constructs.) should be introduced in a more advanced programming course.

In our approach the argument for or against the goto is quieted for the goto never appears in an algorithm. If used in a translation the goto mechanically causes the correct transfer of control specified by the algorithm. Thus program structure and correctness are independent of the goto. Readability is present in the algorithm refinement levels kept with the program for documentation purposes.

REFERENCES

1. Ledgard, H. F. and M. Marcotty, "A Genealogy of Control Structures, *Comm* ACM 18, 11 November, 1975, pp. 629-639.
2. Gries, D., "On Structured Programming—A Reply to Smoliar," *Comm* ACM 17, 11 November, 1974, pp. 655-657.
3. Denning, P. J., "Is 'Structured Programming' any Longer the Right Term?," *SIGPLAN* Notices 9, 11 November, 1974, pp. 4-6.
4. McKeeman, W. M., "On Preventing Programming Languages from Interfering with Programming," IEEE *Transactions on Software Engineering,* 1, 1 March, 1975, pp. 19-26.
5. Flon, L., "On Research Instructured Programming," *SIGPLAN* Notices, 10, 10 October, 1975, pp. 16-17.
6. Keller, R. F., *Constructing Algorithms—An Introduction to Structured Programming,* Kendall/Hunt, Dubuque, Iowa, 1975.
7. Dahl, O. J., E. W. Dijkstra, and C. A. R. Hoare, *Structured Programming,* Academic Press, London and New York, 1972.
8. Wirth, N., "Program Development by Step-Wise Refinement," *Comm* ACM 14, 12 December, 1971, pp. 780-790.
9. Wirth, N., "On the Composition of Well-Structured Programs," ACM *Computing Surveys* 6, 4 December, 1974, pp. 247-259.
10. Wirth, N., *Systematic Programming,* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1973.
11. Knuth, D. E., "Structured Programming with Goto Statements," ACM *Computing Surveys* 6, 4 December, 1974, pp. 261-302.
12. Kernighan, B. W. and P. J. Plauger, "Programming Style: Examples and Counterexamples," ACM *Computing Surveys* 6, 4 December, 1974, pp. 303-319.
13. Bond, R., "Free Form Structured FORTRAN Translator," *SIGPLAN* Notices, 10, 10 October, 1975, pp. 12-15.
14. Gales, L. E., Structured FORTRAN with no Preprocessor," *SIGPLAN* Notices, 10, 10 October, 1975, pp. 17-24.
15. Charmonman, S. and J. L. Wagner, "On Structured Programming in FORTRAN," *SIGNUM* Newsletter, 10, 1 January, 1975, pp. 21-23.
16. O'Neill, D. M., "SFOR—A Precompiler for the Implementation of a FORTRAN-Based Structured Language," *SIGPLAN* Notices, 9, 12 December, 1975, pp. 22-29.
17. Meissner, L. P., "A Compatible 'Structured' Extension to FORTRAN," *SIGPLAN* Notices, 9, 10 October, 1974, pp. 29-36.
18. Kreitzberg, C. B. and B. Shneiderman, *FORTRAN Programming—A Spiral Approach,* Harcourt, Brace, Jovanovich, 1975.

Instructional computer systems for higher education

by CHARLES J. PRENNER
University of California, Berkeley
Berkeley, California

and

ALFRED Z. SPECTOR
Harvard University
Cambridge, Massachusetts

ABSTRACT

In most universities, increased instructional utilization of computers by many departments has been the rule. With the associated diversity of instructional requirements and evolving hardware capabilities, the choice of a proper computer system for this environment has been made more difficult. In this paper, a review of the requirements for such a system is presented and the alternatives are analyzed in light of these requirements. It is shown that non-interactive systems are the least desirable educationally and furthermore, that the economic justification for their use is no longer as strong as in the past. In support of this, a description of an inexpensive interactive system currently in use at Harvard is given.

INTRODUCTION

With an increased need for instructional computing in higher education and tighter university budgets, educational administrators must determine which computer systems provide the highest educational benefits at the least cost. Questions of interactiveness, size and performance must be considered. They can only be answered by an analysis of the user community, of the various possible alternative systems, and of the costs of these systems in light of the conditions on a particular campus.

THE USER GROUPS

In order to determine the desirable attributes for an instructional system, a survey must be made of the various user groups and of their individual requirements. Unlike many systems where essentially one class of end-user is supported, educational systems typically must host a widely variegated user community.* It is convenient to classify the users into four categories: (a) those who utilize the computer for introductory programming courses, (b) users from intermediate and advanced computer science courses, (c) users from non-computer science courses utilizing the computer as an instructional aid or tool, and (d) users consisting of instructors and staff. Each group is analyzed in turn.

Users from introductory computer science courses

This group utilizes the computer to obtain a general understanding of the techniques of computer science. In this category are general interest students, future computer science majors and students from the physical or quantitative social sciences. Although it is true that the individual goals of the students may vary, most students share the same computing requirements. Mainly, they must all cover a large amount of information in a short time. They must also be given a broad and balanced view of the subject matter. Finally, because of their lack of experience, they require especially simple system conventions and a protected environment which provides for easy detection and correction of errors.

A computer system that suits their needs must be capable of supporting a suitably rich set of programming languages as well as providing reasonable conventions for their use. Since introductory students will undoubtedly make many mistakes, easy debugging of programs and clear diagnostics must be provided. Finally, as all neophytes have difficulty in coping with what appears to them to be convoluted conventions, these must be held to a minimum.

* Computer aided instruction, (CAI) *per se* is considered to be a different use of computers and is not considered in this paper.

Users from non-introductory computer science courses

This group requires a wide variety of languages in which to program and the ability to develop, debug and test programs as quickly as possible.

For example, students in this category may have courses in programming language design or implementation where it is necessary for them to obtain experience with the languages. Alternatively, they may be taking a course in algorithms and have frequent need to implement their solutions to problems. At this stage in these students' training it is important for them to be exposed to as many different programming situations as possible; the computer system should not give them a myopic view of their field of study.

Users from non-computer science courses

The members of this user group are learning both to utilize the computer as a tool in their respective fields by carrying out assignments which illustrate basic methods, and also to use library packages to aid them in their work. For example, students in this group may be in an econometrics or statistics class where they are performing sample regressions using library packages, or developing their own software to do this.

This group, being diverse, may yield many requirements. It is composed of future mathematicians, social and physical scientists and various kinds of applications programs are needed for their support. Thus, this group will also require a wide class of programming languages and a powerful library of applications programs.

Usage by instructors and staff *

These users typically have enormous time demands on them and thus require responsive, time-saving systems. It is essential for faculty to have the ability to develop pedagogical and applications libraries quickly and easily.

In terms of other requirements, those of this group may be correlated with the needs of their students. If the needs of the users in the groups described above are met, the needs of this user group will be met as well.

There are three main requirements in an educational computer system which can be distilled from the needs of the four user groups described above: (1) support for a wide variety of programming languages and applications libraries, (2) provision for quick program development and easy debugging, and (3) simple, sensible conventions.

It is important to note that *no* program execution speed requirements have been postulated. This is because repetitious and time-consuming program executions are not commonly found on the usual instructional university machine. While it is true that some students will utilize the computer to solve problems using library applications programs, the tasks tend to be small since the computer problems assigned in most courses are the minimal sized examples which demonstrate the desired principles or techniques.

Thus, system throughput is not critically dependent upon program execution speed. Rather, it depends on *program development efficiency* because constant program development is the activity which best characterizes the university instructional computer system.

ALTERNATIVE SYSTEMS

The needs of the various user groups may be met in different ways. The users may be supported on a variety of machines, each offering distinct facilities. For example, there may be the need for some special purpose computers to act as real-time laboratory aids or to be used stand-alone in an operating system course. However, the needs of most users will be supported either on non-interactive systems (batch) or on interactive time-sharing systems. It is to these systems that we next turn our attention.

Non-interactive systems

The non-interactive, or batch, machines are usually fast and capable of executing the largest programs. However, execution-speed/size efficiency is not obtained without a price. Batch machines are inefficient as a tool for program development, especially in an environment in which a large support staff is unavailable. In addition, they are unusable in situations where interactiveness is a requirement. Finally, in some sense, they are an improper model of a computer to present to the student, especially the neophyte.

As a program development tool for those learning to utilize computers, batch machines are very wasteful of both machine time and more importantly, human time. Associated with batch machines is the notorious *batch cycle*. This consists of the four step process which must be repeated numerous times while writing and debugging programs: (1) the program must be written or corrections made, (2) program (and job control cards) or corrections must be keypunched, (3) the card deck must be submitted and (4) the print-out must be awaited. This cycle is especially onerous in an instructional environment because students will make large numbers of mistakes, both conceptual and typographical. The latter mistakes are compounded by the fact that students *are not professional keypunchers*. Thus, the batch cycle will be repeated an excessive number of times even in the course of easy program development. This ties up both operator time, student time,

* The reader should note: that this paper is concerned only with instructional uses of computers in the university. If it is desirable to have the faculty and staff utilize the instructional machine for their own research purposes, the requirements can grow considerably.

and burdens the system with a constant influx of trivial tasks.

Further problems with program development are associated with the fact that debugging aids are necessarily static on a batch machine and therefore, they are not nearly as flexible as the dynamic aids on an interactive system. Again, this is an especially grave deficiency in a university environment due to the large number of errors that students will make.

In addition, the very fact that batch machines are non-interactive means that restrictions are placed on the flexibility of the system. Certain kinds of computer usage cannot occur at all and other kinds not very easily. For example, the batch machine is not usable in a laboratory situation where a student wishes to quickly analyze data necessary for an ongoing experiment. In other situations, it may be very desirable, though not strictly necessary, for the user to dynamically view the results of a program and be able to interact continuously with it.

The final disadvantage of batch systems is a psychological one. In using a batch machine, the student must necessarily learn of the computer as a static machine. The student sees the computer as a monolith which can talk only *at* him, never *to* him. Undoubtedly, this is a bad view to give to the student, especially the introductory student who may be uncomfortable with computers in the first place.

In summary, batch machines have the capability of executing large programs efficiently but provide an inadequate program development medium for students. They are restrictive in the kinds of tasks in which they are useful and are a poor model to present to neophytes. It should be noted, in light of the conclusions on the needs of the user groups, that the ability of batch systems to execute large programs efficiently is not extremely valuable on an instructional system. It is a capability which is not necessary for this environment.

Thus, the batch computing system does not fulfill the requirements of the user groups and is not a good choice as the basis of an educational computing system. This is not to say that batch machines do not have a place on campus. In the bulk of computer usage, efficient machine utilization is vital and batch machines can be used effectively. However, in contexts where there is almost constant program development and great diversity of needs, batch computers are not suitable.

There has been a variant on the batch machine which has given to the batch system some of the attributes of interactive systems. Systems like Wylber[1] have provided some degree of interactiveness and at the very least, allow the user to dispense with key-punching. To the extent that the system can provide interactive facilities, the system can meet the requirements described earlier.

Interactive systems

Although, interactive computing is an old concept in educational systems,[2] it is still rarely utilized.[3] The machines which are utilized for interactive computing are very diverse and encompass great differences in flexibility, machine cost, performance, and even interactiveness. Some systems have the capability of running in a batch emulation mode, while others provide no such opportunity. Some systems utilize only one language while others support many diverse languages. In this section, we classify the different types into four categories: (1) large scale time-sharing systems, (2) traditional mini-computer systems, (3) large mini-computer systems and (4) networked systems.

The large scale time-sharing systems are characterized by *many* simultaneous users, relatively large size and speed, and high flexiblity. In fact, the systems tend to be so large that many universities have too little computing to fully utilize a whole machine. Thus, the system must be shared with others. This can lead to high communication costs and other problems associated with an environment in which there is not complete control over the computing resource. However, as will be seen, from strict performance criteria, these machines do an excellent job of satisfying the educational requirements.

Since the large time-sharing machines are fast and possess large address spaces, they can run the most complex instructional programs. In addition, the large address space allows the use of many different language systems as well as the use of large flexible software designed to simplify the program development task. Furthermore, they can allow for any amount of interactiveness including on-line editing, dynamic program debugging, and graphics. Some systems even provide a batch mode in which more complex, time-consuming tasks can be run.

Of course, these machines do not have the efficiency advantages of the batch machines in raw execution power but as has been argued, this is wasteful in an educational (non-research) environment. In fact, it is argued here that even these large time-sharing machines, like the large batch machines, have built into them an amount of execution efficiency beyond that which is required in the instructional environment.

Traditional mini-computer systems have tried to provide the services of their large counterparts but with more restrictive environments and less powerful processors. Many systems have only a single language available, typically BASIC, and can handle between 5 and 15 users simultaneously. These machines have many of the advantages of the large time-sharing machines and thus satisfy some of the requirements of the user groups. However, they are not powerful enough to meet the flexibility requirements.

These machines are interactive and thus allow ease of debugging, instantaneous turn-around, and quick program type-in. However, due to both the restrictive-ness of the operating systems which run on them, the speed of the CPU and the restricted main memory size, they cannot support languages powerful enough for many users. For example, the "BASIC only" systems are suited for some kinds of programming but certainly cannot fulfill the requirements for an intermediate course in computer science.

Thus, these machines do not fulfill the requirements of the user groups because of their low flexibility. They are not a good choice for a university system.

Large scale mini-computer systems are based on the great advances in hardware technology that have allowed for the creation of much larger and more powerful mini-computers. Although many of these machines have limited processing speed and memory restrictions, they are much more powerful than the mini-computers which were available only a few years ago. We contend that one or more of these machines are sufficiently powerful to provide for almost all of the educational computing needs for a university. Typically, each can serve between 20 and 50 users, each user having the ability to perform a wide variety of commands and utilize a wide variety of languages.

The reason that this can be achieved is that most requests for service in the instructional environment are trivial ones which have correspondingly little need for raw computing power. Thus, a relatively large number of users can be supported on a system of only moderate power. Occasionally, some students' requests will be beyond the power of the system. But, in the authors' experience, the number of such requests is typically small. For these students, alternative arrangements can be made.

An approach that may be used in the future is to connect a number of large-scale mini-computers directly to a powerful central processor in a hierarchical network. If any of the mini-computers become overloaded, they may send some users' requests to the more powerful machine to be executed. In this way, reasonable response time can be maintained on the small machines. In addition, the small number of users whose computations would normally overflow the large-scale mini-computer systems may be accommodated.[4]

Thus, of the time-sharing approaches, only the use of the more traditional mini-computer based system does not meet the requirements of the user groups. If the university administrator is concerned with performance alone, the large scale time-sharing system would be his choice. However, as the cost of such a system may exceed the funds available, it may be necessary to choose among the other alternatives.

COSTING THE VARIOUS SYSTEMS

The university administrator, after taking into consideration the merits of the various kinds of computer systems, as discussed in the previous section, must choose between systems based on their educational cost-effectiveness. This determination is exceptionally difficult[5] and is a two part analysis. First, the cost per standardized resource unit must be determined. Second, the educational benefit per such unit must be evaluated.

If educational effectiveness per unit resource is not considered, it is possible for a computer system to appear to be less expensive than it actually is. For example, it is easily conceivable (and even expected) that a student may require more runs on a batch system in order to debug and test a program than on an interactive system. Thus, even if the batch system were to have a smaller cost per job executed, the overall cost of program development could be higher.

The difficulties in determining cost per resource unit are also considerable. The construction of a standardized unit with which to compare systems is in itself a difficult task. This can be seen to be extremely hard when comparing batch and interactive systems. However, even on a single system, it is difficult to present clearly the assumptions that go into the determination of some costs.

Thus, whenever a definitive assessment of the benefits and costs of a given system is made, it should be kept in mind that the task is extraordinarily difficult and that the results of any studies should be viewed cautiously. Finally, explicit specification of the numerous assumptions used in any study are required if the study is to have any meaning.

The authors' experiences have been with educational time-sharing systems. However, it has become clear to them that even when comparing and attempting to describe the attributes of systems in this category alone, the number of items which must be considered is very great with respect to the description of the costs of a given system, and of its attributes.

For example, on interactive systems, it is common to use *connect cost per hour* as the yard-stick of cost. This is usually defined by the following fraction:

$$\frac{\text{Initial Cost/Yr.} + \text{Operating Cost/Yr.}}{\#\ \text{Connect Hours/Yr.}}$$

where

$$\text{Initial Cost/Yr.} = \frac{\text{Hardware} + \text{Software Cost}}{\#\ \text{Years of Amortization}}$$

But, it should be immediately seen that this fraction can vary substantially, perhaps by an order of magnitude depending upon the interpretation of the various terms. For example, are actual yearly connect hours utilized, or some "reasonable" figure which could be assumed to be the maximum (or minimum) number available? Or is the "reasonable" number of connect hours based on raw hardware limitations or upon the maximum number of users which can obtain some reasonable response time? There are, of course, many more questions which could be asked.

It should also be noted that the cost from university to university can vary greatly. Perhaps, the university can support some percentage of the cost of the instructional machine by utilizing it for some research or some grants can be obtained to cover certain parts of its development. Also, there can be wide variations in the cost of the hardware from institution to institution.

Because of these difficulties, no attempt is made to provide a detailed cost comparison of various time-sharing systems. Unfortunately, this would be the only way to demonstrate that large mini-computer based systems are cost effective. But this would be difficult, in general, since there will be instances where circumstances particular to a given institution make a different kind of system more cost-effective. Instead, a detailed case study of the Harvard undergraduate time-sharing system is presented.[6,7] This system is one in which the authors have extensive experience and one which has been running long enough to provide comprehensive data. In the particular situation in which it is used, it appears to provide for extremely inexpensive instructional computing. It also appears likely that there might be many comparable situations.

A LARGE MINI-COMPUTER BASED SYSTEM EXAMINED

The present Harvard undergraduate time-sharing system (HRSTS) is utilized in the support of the computing requirements of most undergraduate courses, some graduate courses and a limited amount of research usage. Among others, courses in applied mathematics, economics, mathematics, engineering, music, and even Arabic utilize the system. The hardware used is a Digital Equipment Corporation PDP11/45 processor,[8] 240K bytes of core memory, a fixed head disk for swapping, 116 Megabytes of on-line disk storage for files, a high speed printer, card reader, paper tape reader/punch, 2 DEC-tape drives and about 28 terminal ports.

The operating system used is a modified version of Bell Laboratory's UNIX system.[9] The language processors currently in use on the system are 2 assemblers, 2 text editors, numerous high level languages including BASIC, FORTRAN, C,[10] PPL,[11] ECL,[12] and LISP as well as a wide variety of utility programs. The user command interpreter was specially designed for ease of use based on past experience with other time-sharing systems.

The system is operational 23 hours a day, 7 days a week with periodic maintenance disturbing this schedule somewhat. The load conditions vary from semester to semester. During the first semester, PPL is used heavily by the 400 students of Harvard's general introductory computer course. During the second semester, the assembler and LISP are most heavily used due to an intensive introductory computer course for more advanced students. During both semesters, FORTRAN and BASIC are utilized extensively by students dealing with numerical methods and due to continual system development, the language C is also highly utilized.

The system is operated by one full-time administrator and numerous "terminal watchers." The latter are students who are hired by the university to act as combination operators, programming assistants, and systems programmers. During most of the day, they are available. This is especially valuable for the large number of beginning users.

The use of terminal watchers highlights one of the cost advantages of the university owning its own system. The cost of the terminal watchers is low due to the lower cost of student wages and the fact that a payment from the university to a student is really an intra-university transfer payment which is not too costly to the university and very beneficial to the student.

The present operating costs for the system are sketched in Table I. It should be noted that the budget is inflated by four items: (1) the cost of the terminal watchers, many of whom would not be necessary if it were desirable to have only operators on duty during usual hours or if such personnel could be recruited from the teaching staffs of the courses using the system, (2) the high cost of having rented, high speed video display terminals, (3) the high cost of maintenance contracts with the various manufacturers (as opposed to in-house maintenance), and (4) the cost of having dial-up lines on the system. The latter cost shows that communication costs are a very important consideration in the overall costs of a time-sharing system. In a different environment, the costs could be reduced significantly.

The capital costs for the system have been amortized over a three year basis and come to approximately $50,000 per year. The amortization period is not based on the life expectancy of the system for the system is now 18 months old and there is every expectation that it will last considerably longer than an additional 18 months.

The system usage, over the Spring and Fall semesters of 1975, is shown in Table II. As can be seen, the total number of connect hours utilized during the year is 43,500. It is the belief of most people associated with the operation of that system that this number of connect hours is close to the maximum that the system can support.

Based on the *actual* connect hours of the year 1975, the connect cost per hour is approximately $3.20. It

TABLE I—Approximate Operating Costs for HRSTS

Item/Description	Amount
Terminal Watchers & Other Salaries	$41000.
Supplies	$ 5000.
Telecommunications	$ 9000.
Terminal Rental	$22000.
Maintenance	$13000.
Total	$90000.

TABLE II—Total Connect Hour Usage of HRSTS during 1975 by major usage category

Category	Spring	Summer	Fall*
Applied Mathematics	7685	—	300
Arabic	99	149	400
Biology	232	110	1100
Chemistry	1492	215	100
Economics	29	55	200
Engineering	496	115	300
Grad. School of Design	1028	10	1500
Independent Usage	1053	488	1000
Mathematics	753	93	300
Natural Sciences	297	28	8100
Physics	1159	575	300
Social Sciences	346	90	100
Statistics	362	11	500
Research	336	135	200
Sys Work/Term. Watchers	3288	1995	4000
Other	450	1270	800
Totals	19105	5339	19200

* projected based on Dec. 20, 1975 totals.

should be noted that this figure does not include an extraordinary once-only development cost of $50,000 for some of the system software. If the reader is concerned with duplication cost of the system, this figure should not be taken into account since most of the software is now freely available to other educational institutions.

HRSTS COSTS IN PERSPECTIVE

The determination of the relation of these costs at Harvard to real costs in other situations must necessarily take into account the following consideration.

(1) *Usage*—The system at Harvard is highly used, often in the very early hours of the morning. The impact of this is substantial on the connect cost per hour figure cited above.

(2) *Communication Costs*—At Harvard, there are many dial-up lines available. If the terminals are in one or more centralized locations on campus, the number of dial-up lines can be reduced substantially.

(3) *Overhead*—The costs given do not include the cost of utilities, room space nor the *de facto* use of some of the administrative structure of the Harvard University Science Center.

(4) *Operator Attendance*—The cost of operator coverage could vary substantially at other installations depending upon the availability of students to handle this function and the amount of coverage desired. It could conceivably be made the responsibility of teaching assistants.

(5) *Peripheral Hardware*—Considerable peripheral hardware exists at Harvard that might not be needed elsewhere. However, additional items might be necessary if real-time or graphics applications are desired.

(6) *Application Programming Costs*—These are specifically not included in the connect cost per hour figure cited above except to the extent that they are handled by the terminal watchers. In some situations, it might make sense to include these costs.

(7) *Amortization*—The period at Harvard is three years. It might be more reasonably set at five to seven years.

(8) *Specialized System Software*—Most of the software used at Harvard may be obtained free by other universities. However, the development of new software might be required for certain situations.

(9) *Hardware Advances*—The compatible Digital Equipment Corporation PDP11/70 processor[13] can support at least twice as many users without very great increases in cost.

It is possible to reduce the cost per connect hour to $0.50 per connect hour under favorable circumstances (no dial-up lines, operator coverage by teaching assistants, local maintenance, longer amortization, and purchased low cost terminals). Since most environments will differ somewhat from this extreme, the expected costs will be in the $1.00 to $3.00 range.

CONCLUSIONS

The low cost of HRSTS lies in the fact that the hardware is tailored to the needs of the users. Such systems utilizing large mini-computer systems, tend to be just powerful enough to support the vast majority of the users' demands. In the future, it is conceivable that a hierarchical network based on machines of differing power will be the logical extension of this. Easy computing tasks will be serviced by the simplest and lowest cost machines with the more diffcult tasks being passed on to larger machines.

This division will allow the cheapest machines to be utilized in most instances and the more complex and expensive machines will be saved for the limited amount of overflow. This approach will allow the benefits of the largest time-sharing systems to be coupled with the very low costs of the mini-computer based systems.

Today, a large mini-computer based system, not unlike Harvard's, can offer very low cost, flexible computing that satisfies most requirements for an educational system. The services provided by such systems are much more suited to the needs of the university user community than are those provided by batch or single language systems. Although not as powerful as the large scale time-sharing system, they are sufficiently powerful to satisfy the needs of most users. Furthermore, such systems can be run with very low cost, with connect costs in the $0.50 to $3.00 range.

REFERENCES

1. Fajman, R., J. Borzgelt: "Wylber: An Interactive Text Editing and Remote Job Entry System," *CACM*, Vol. 16, No. 5, May, 1973.

2. Kemeny, John G. and Thomas E. Kurtz, "Dartmouth Time-Sharing," *Science*, Vol. 162, No. 3850, October 11, 1968.
3. McCracken, John, "Is There a FORTRAN in Your Future," *Datamation*, May, 1973.
4. Prenner, Charles J. and J. P. Buzen, *Proposal For Research on Hierarchical Computing*, Ctr. for Res. in Computing Tech., Harvard University.
5. Austin, John E., *Costing Computer Aided Instruction*, Ibid.
6. *HRSTS Terminal Users Guide*, Harvard University Science Center, August 1975.
7. Prenner, Charles J., Using Unix in an Instructional Environment, *Proc. COMPCON* 1976.
8. *PDP11/45 Processor Handbook*, Digital Equipment Corporation, 1973.
9. Ritchie, Dennis M. and Ken Thompson, "The Unix Time-Sharing System," *CACM*, Vol. 17, No. 7, July 1974.
10. Ritchie, Dennis M., *C Reference Manual*, Bell Laboratories.
11. Taft, Edward A. and Thomas A. Standish, *PPL User's Manual*, Ctr. for Res. in Computing Tech., Harvard University, TR 21-74.
12. Wegbreit, Ben E., "The ECL Programming System," *Proc. FJCC* 1971.
13. *PDP11/70 Processor Handbook*, Digital Equipment Corporation, 1975.

ADP training systems—Organization-wide training for increased productivity

by JACK L. STONE and ALEXANDER P. GRANT
Computer Education International, Inc.
Washington, D.C.

ABSTRACT

In most organizations with medium to large computer centers, management attention is primarily focused on technological gains as the major method for improving data processing services and utilization. However, computer system productivity can also be increased through effective organization-wide training.

To improve on training currently available to most organizations—OJT, standard courses provided by the hardware manufacturer or independent suppliers, and self-study packages—a systems approach to training is suggested, named the ADP Training System (ATS). This approach employs proven systems methodology to plan, develop, and implement fully documented ADP training to improve job performance of all personnel who manage, operate, or use the computer center facilities and services. An ATS is tailored to meet specific needs of the organization.

ATS planning includes determination of organization needs, available resources, and program constraints. It documents objectives, scope, costs and master schedules.

ATS development includes course definitions, development schedules, and course materials development. It provides implementation schedules, instructor's guides, student materials, and support materials.

ATS implementation includes administration, execution of courses, evaluation and redirection.

After a discussion of ATS concepts, a case history of an ATS program which involved the authors is presented.

It is concluded that an ADP Training System provides management the planning and control capability to implement organization-wide training for new systems and technology on a cost-justifiable basis.

INTRODUCTION

As the use of automated data processing for commercial applications has become more and more widespread, it has become increasingly worthwhile to examine those factors that influence the cost/effectiveness of ADP systems. This paper addresses one such area: organization-wide training for increased productivity. It advocates the application of total system concepts to the development and implementation of training programs and presents an example of the use of those concepts in an actual situation.

In most organizations with medium to large scale computer centers, personnel costs exceed the costs for computing equipment and software; nevertheless, management attention continues to be focused on technological gains as the primary method for improving data processing services and utilization. Granting that processing costs can be substantially reduced with improved hardware technology, it is also true that the over-all effectiveness of complex data processing systems can be improved through effective organization-wide training.

However, in many, if not most, organizations, computer center hardware and software have surpassed the capabilities of existing programs to train personnel to manage, operate and use this technology. In many organizations ADP training is only available on an ad hoc basis, often unplanned, and provided as on-the-job training, instruction at a computer manufacturer's or independent supplier's education center, and self-study of textual or audiovisual packages. In short in many organizations, ADP training tends to be episodic, fragmented and directed toward the satisfaction of unanticipated needs.

After extensive experience as vendors of training services to many organizations such as those noted above, the authors have developed and implemented a systems approach to training that they call ADP Training Systems. The ADP Training Systems method employs proven planning, systems and training methodologies to produce a fully documented ADP training program that is directed toward improvement of on-the-job performance of all personnel who manage, operate or use the products of an organization's automated data processing system. Each ADP Training System is developed for a specific organization and has as its major objective satisfaction of that organization's training needs in a coherent, timely and economic manner.

An ADP Training System is an organization-wide program for the training or re-training of personnel to meet new or changing job requirements that are occasioned by changes in the scope or level of automated data processing technology available to the organization. It is concerned with a broad range of personnel, including: top executives; middle management; project administrators and supervisors; users; systems analysts; applications and systems programmers; and operations personnel. It is characterized by an orderly, planned process of installing an effective operational training capability to support all of the training needs of the organization in an economic, efficient manner.

The scope, complexity and depth of penetration of a specific ADP Training System is dependent on the size, complexity and needs of the specific organization for which it is intended. In any case, to some extent it always includes three functions:

(1) Planning;
(2) Development;
(3) Implementation.

Planning—This function includes determination of: the organization's training needs; the resources available to support a training program; and the considerations and constraints that must be taken into account in developing and implementing the ADP Training System. The function's outputs are a set of planning documents that define: the objectives of the ATS; its scope and content; its probable cost for development and operations; and the estimated timeframe for development and operation of the ATS.

Development—This function includes: definition of the individual training courses required; setting development schedules; selection or development of course materials. The function's outputs are: a program implementation schedule; instructor's guides; student materials; class schedules, class announcements and other ancillary materials.

Implementation—This function includes: administration and execution of individual courses; evaluation of students, instruction, and overall system operation; modification or re-direction of the system as needed.

PLANNING FOR AN ADP TRAINING SYSTEM

In order to do a creditable job of planning for an ADP Training System, it is necessary that the planner have a reasonable understanding of the goals, activities and organization of the company or agency for which the ATS is intended. The normal investigative and analytical techniques used in any system study are applicable to this problem. Once the organization itself is understood, the ATS planner can move toward definition and documentation of the organization's training needs and objectives.

The fundamental goal of training in any organization is, or should be, to improve performance on the job. Specific, more detailed, objectives flow from this goal. Any departure from this goal will necessarily result in a decrease in the cost/effectiveness of the system. Courses of an academic nature, while useful in the long range, do not necessarily support the organization's operational objectives and are, therefore, difficult to justify. It is more cost/effective to encourage individuals with academic interests to pursue course work at a local college or university.

Training, within the operational organization, for both technical and non-technical personnel should be oriented to specific job functions and particular ADP applications. With this strategy, both management and employees can better understand the value of ADP training: management can more easily evaluate the financial investment in terms of increased personnel productivity and the ability of personnel to handle increased responsibilities; employees can more readily accept the training as a vehicle for improved performance on the job, self-development, career enlargement, and professional growth.

In addition to satisfying the basic goal of improved job performance, an ADP Training System should be tailored to meet the special needs of the organization, including: selection of topics to relate the instruction directly to the installed or anticipated computer configuration; development of computer-oriented student problems; off-hour or part-day scheduling of classes to accommodate the regular workload of the students; and, when appropriate, the development of case studies based on actual experiences at the installation.

Planning for an ATS should also include consideration of factors that influence training costs. In the authors' experience, appreciable cost savings can be effected through implementation of training programs on-site which allows use of installation-owned classrooms and equipment for instructional purposes and eliminates student travel expense and idle time. For an on-site course, $50 per student for each instructional day* appears to be an achievable goal for groups of 12 or more. This number should be contrasted with $75 to $150 a day per student at typical classes held by manufacturers and producers of commercial seminars.

Some other considerations that bear on the development of a plan for satisfying the training requirements of an organization faced with technological change, relate to the training curriculum. First, training must be implemented for all personnel categories involved with the computing center or using its products. Such training may range from a two-hour seminar for top executives to a six month case study in systems analysis and design. Training should not be limited to tech-

* Six classroom hours

nical professionals, i.e., analysts and programmers, but should be extended to technician-level personnel, i.e., coding clerks, computer operators, tape librarians and so on, and to users, managers and executives. In addition, each training course should be related to a specific personnel category. Courses should be organized as a progressive sequence of formal classroom instruction, interleaved with practical OJT assignments as appropriate. Refresher and review courses of short duration should be planned to provide technical updates, answer questions regarding current operations and future plans, and to minimize the normal tendency of individuals to drift away from established systems and procedures.

The outputs of this developmental phase consist of a set of planning documents, including:

(1) A statement of the objectives of the ADP Training System;
(2) A description of the course content of the ADP Training System;
(3) A summary estimate of the resources required for the ATS including a cost estimate for each major class of resource;
(4) A proposed schedule for the ATS;
(5) An outline and summary for each course including planned OJT segments. The documentation for each course should include:
    —The behavioral objectives of the course.
    —A profile of the typical target participant including: prerequisite training and experience; current work assignment; and proposed work or job functions to be assigned at the completion of the course.
    —A topic outline of the course content.
    —A description of the proposed instructional methodologies for the course.
    —A description of student materials.
    —A proposed schedule for the course.
    —An estimate of the resources required for the course including a cost estimate for each major class of resources;
(6) A plan and schedule for development and implementation of the ADP Training System.

The completed ATS planning package is presented to management for review and concurrence. Any needed revisions to the plan should be made before proceeding to the next developmental phase.

DEVELOPING AN ADP TRAINING SYSTEM

Upon acceptance by management of the ATS plan, the ADP Training System package may be developed and documented. The major outputs from this activity are:

(1) The ADP Training System Catalog;
(2) Course announcements;
(3) Structured training course packages.

The ADP Training System Catalog is an edited version of the approved ATS plan. It is in a form suitable for distribution to prospective students. Publication and distribution should be in accordance with normal policies of the organization. Course announcements are extracted from the catalog as needed.

The structured training course package consists of two major components: the instructor's guide with its supporting materials and the appropriate student materials. The basic instructor's guide is composed of:

(1) A narrative exposition of behavioral objectives, content and instructional methodologies;
(2) A course schedule;
(3) A listing of student materials;
(4) A set of lesson plans which set forth a detailed description of: instructor activities and instructional aids; the instructional content of the unit; lesson objectives; and student activities.

Appropriate student materials are developed or selected and procured, and are coordinated with the instructor's guide.

IMPLEMENTING AN ADP TRAINING SYSTEM

Implementation of an ADP Training System consists of system administration and course presentation. The system administrator need not be an educator or teacher, but he should have a firm grasp of the objectives of training and of good administrative practice. His functions are to: select or approve instructors; announce courses in a timely fashion; administer the student selection process; assure the availability of needed facilities, materials and equipment; evaluate overall system operation; and prepare appropriate management reports.

Instructors should be selected from professionals in the data processing field who have achieved a reasonable balance among technical experience, instructional experience and communication skills. Experience has shown that instruction in technical subjects requires an unusually high degree of instructional talent which includes not only a good background in the subject matter, but also a high level of empathy with the students.

It is important that the ADP Training System and its constituent courses be properly announced and supported to insure that all prospective students and their supervisors are aware of the available training opportunities and can have the time necessary to plan for their attendance. Elements of the announcement strategy include: letters from the chief executive requesting support of the training function by line management; the early distribution of the ADP Training System Catalog; briefing sessions for prospective students to explain the program or courses and to answer their questions; course announcements through bulletins or other communication media in the organization.

Each prospective student for each course should be screened to assure that he satisfies course prerequisites and that his post-training assignment will be appropriate. In addition, the system administrator should assure that each student's current work schedule has been adjusted to allow class attendance.

Effective training requires that students receive all documentation appropriate to the subject matter. For example, students training in hardware/software technology should receive personal copies of appropriate technical manuals; students in introductory courses should receive text books written at levels commensurate with their capabilities, e.g., executives should receive texts written at a general level. Other student materials such as charts, diagrams, templates and the like should be distributed as needed.

Audiovisual materials can be valuable instructional aids. Such aids reinforce textual and oral presentations and provide an enlivening change of pace. However, such materials must be integrated with other course activity: each audiovisual segment should be both introduced, and followed up by the instructor. The system administrator's responsibility in regard to audiovisual support is to assure that both materials and any required equipment are available when and where needed.

It is important that administration of the ATS be handled just as any other project; therefore, the training administrator should prepare periodic management reports regarding overall performance and adherence to budget.

In the authors' experience, classroom operation is most successful when well structured and tightly disciplined. In this approach, daily quizzes based on outside reading assignments and the previous day's class work provide a sharp focus on the instructional content of the course. Each class session starts with a quiz which is then corrected immediately in a process that provides the framework for discussion and clarification of the previous session and reading assignment. The extensive testing is used primarily to reinforce learning and to provide the students immediate goals for learning. It is only secondarily used for evaluating student progress.

As appropriate, many instructional methodologies are used, including: oral presentations by the instructor; audiovisual materials; laboratory exercises and workshops; and outside reading or project assignments. The particular methodologies used in a specific course are dependent on course content, course scope, and the type and level of persons being trained.

A CASE HISTORY

This case history involves the ADP Training System that was established for a large federal agency that was upgrading its computer capability. Its second generation equipment was being replaced by a large third generation computer with advanced software support and remote terminal and RJE capability. Implementation of the new system was contracted to an industrial firm who provided on-site technical assistance, system development, computer center management and training. The implementation plan extends over a five year period.

In the early years, the training plan was to provide basic technological training and to develop an in-house training capability. It is intended that the contractor will gradually phase out as agency personnel become capable of picking up responsibility. At the end of five years, it is intended that the agency will be managing all aspects of its operational and training program.

The original plan called for training programs to be set up for approximately 100 programmers and analysts, 50 computer operators and support personnel, 350 terminal users and operators, and 150 executive and middle management personnel. The initial training period extended over approximately 18 months ending in December, 1975. The original plan called for approximately 500 days of instruction over this period. The authors' company participated in a substantial portion of the technical training of personnel.

Although the program was reasonably successful, some areas of possible improvement are documented for future reference. Such comments relate only to segments of the program that were produced by the authors.

At an early date, a master plan and schedule was produced by the prime contractor and accepted by the agency's management. This plan had a reasonable array of courses scheduled in progression from basic to more advanced training. The authors' company was engaged as a subcontractor to produce courses to meet the plan's specifications.

These courses dealt with four training areas:

(1) Systems programming;
(2) Programming languages;
(3) Operator training;
(4) Terminal user training.

The systems programming training consisted of thirty instructional days presented over a four month period. This schedule permitted students to attend class part-time and to perform their regular assignments during out of class hours. The instructional goals were met, but not without difficulty. The students' job assignments imposed a heavier and heavier burden as the application conversion program proceeded. Additionally, the four month time frame tended to dilute the instructional impact of the course. A better result would have been attained by organizing the content of the course into two or three courses of shorter duration.

The courses in COBOL, FORTRAN and JCL were only partially successful. Although the courses were announced with appropriate prerequisites, no significant controls were exercised over attendance. As a result many students did not meet the prerequisites and had great difficulty with the course. Additionally, many students lacked any real job-connected objective to be met by this training, that is, neither current nor prospective job assignments required use of the knowledge and skills to be attained by successful completion of training. The presence of a substantial number of such individuals in these courses seriously compromised the effectiveness of the training.

The objective of the three day operator training course was to indoctrinate second generation computer operators and production control personnel with third generation ADP system concepts. The presence of production control personnel in this course presented some difficulties since, in contrast to the computer operators, they had little prior ADP training. In any case, training for computer operations personnel is a complex problem, because such personnel represent a wide range of interests, backgrounds and capabilities, and frequently lack the motivation and study skills needed for mastery of complex technical subjects.

Terminal user training, presented in a four day course, was quite successful. The course, intended to introduce administrators and professional personnel to the use of remote terminals to access an inter-active programming system and an on-line data base management system, was of sufficiently short duration so that student interest was easily maintained. Since this was an introductory course, the divergent backgrounds and objectives of the students did not constitute a severe problem; however, in the future, the course outcomes could be improved by grouping individuals with similar interests into separate classes. In this way, both the instructor's presentation and the workshop sessions which are an integral and important part of this course could be more specific to each group's interests.

Many of the problems encountered in producing the courses discussed above, flowed directly from a lack of strong, sustained management interest. At the beginning of the program it was recognized that an inhouse training capability was required. Four full-time agency people were initially assigned to the program; however, after a short time, management decided that those individuals were needed elsewhere and the inhouse training function was abandoned.

This shortfall in management interest was strongly felt, since the training courses that made up the program were provided by four different organizations. Lacking strong program administration, each provider interpreted the plan in its own way and handled its part of the program in its own style. The overall effect was that the benefits realized from this ADP Training System were less than might have been reasonably expected.

Nonetheless, much about the ADP Training System was right: the organization's training needs and objectives were properly defined; a realistic and effective training plan was documented; a large number of good training courses were developed and implemented. The shortfalls in the student selection process and in program administration should not be allowed to obscure the very real benefits produced by this ATS.

Teaching art through computer graphics

by JOSEPH SCALA
Syracuse University
Syracuse, New York

ABSTRACT

The computer is becoming an important tool for the production of works of art. Art students are becoming increasingly interested in tapping the potential of the computer in the area of static and animated graphics. This paper deals with my personal experience in teaching art through computer graphics, at Syracuse University, to art majors and students from other academic disciplines.

A very exciting experience for me in the past two years has been establishing a computer art division within the department of Experimental Studies in the College of Visual and Performing Arts at Syracuse University and teaching a number of very enthusiastic students from a number of different academic backgrounds.

Students have access to the Syracuse University Computer Center, with the following hardware: an IBM 370 computer, the DEC-10 system with 128K core and a VB10-C display, two high speed line printers, six ADDS CRT terminals, 100 DEC writer terminals, a 14 inch Cal-Comp drum plotter and a 33″ Cal-Comp drum plotter with three pens. Students may use these facilities in conjunction with the university's video, film and animation studios. Art and science students have accepted the computer as a viable tool for the creation of art. Not all students have reacted positively, but the vast majority of students discovered a positively fantastic environment to create in, bringing together the blissful marriage of art and technology.

The computer may be the universal tool, not only a device for scientific and engineering progress, but a tool for artistic progress as well. The computer is malleable, like clay, making it a perfectly applicable medium for manipulation by artistic minds and hands. Naturally, the digital computer alone cannot create art. Artists need hardware and software they can understand. Fortunately, there have been a number of ingenious software systems designed for them. The two principal systems that most of my students use at this time are MINI-EXPLOR and PSPLAT.

MINI-EXPLOR was created by Ken Knowlton and the name EXPLOR stands for the production of images from **E**xplicit **P**atterns, **L**ocal **O**perations and **R**andomness. MINI-EXPLOR is a FORTRAN coded version of the EXPLOR language which allows the student to become familiar with many of the FORTRAN language subprograms and other FORTRAN details from a graphical point of view. The paper output is 132 spots wide by 140 lines long. The display output is 1024 points x 1024 points using run length vectors which are multiply interlaced.* Each individual image can be produced as many times as the artist wants and each individual point or area of points can be changed explicitly or changed randomly by the student's program. There are also subroutines which allow for growth and combination patterns. The graphic output produced on unlined white paper is immediately suitable for framing and editions can be run off in a matter of minutes. In addition, other media such as paint, pastels, different colored line printer ribbons and photography can be combined with the graphic output to bring the images into yet another visual plane. Taking a photograph of the printer output image, developing the negative, making a kodalith from the negative and applying that to the production of a photo-silk screen, photo-lithograph or photo-etching have produced dramatic and colorful prints.

Images generated on the display can be filmed in single frame to produce an animated sequence. The film can then be placed in a film chain in the television studio and these images can be manipulated through the special effects generator, resulting in a color video tape which can then be kinescoped if desired.

Some students prefer to generate sequential images on paper output and single frame each image in our animation studio using the Oxberry animation stand. The advantage of this technique is that the artist can add hand-done animation to the computer graphic, adding other media and images as well as color, giving surprisingly unusual results.

MINI-EXPLOR produces output in four level grey scale. The total grey scale in a single image is easily

* Rather than the conventional video scan-line ratio of 1 to 2, this system uses a ratio of 1 to n.

separated into three images containing one grey level each, with a simple program, allowing the student to add optical color through filmic techniques to his or her animated sequence.

In addition, MINI–EXPLOR designs are beautifully suited for making rugs, weavings, textiles, needlepoints, collages, and relief sculpture.

I have found that in the beginning of my course those students who have a background in FORTRAN and mathematics, usually the computer science and engineering students, are able to produce more interesting designs with greater facility than the art students, who usually have not more than one year of high school algebra and no programming experience. Eventually, there is a leveling off point and then art students begin to excel.

In the second semester of the course art students reach a point where they can begin to use the language effectively enough to establish a personal style. With MINI-EXPLOR this often requires the creation of the student's own subroutines and the delineation of a clear and purposeful aesthetic concept. Even then the imposed stylistic format of the software gives the work a certain 'EXPLOR' character but this becomes less important if the concept is strong. The hardware also contributes a stylistic character to the output but the artist is free to mix media and in this way, if he or she wishes, can effectively eliminate any reference to the software or the hardware to suit the aesthetic concept.

Two other factors that students can capitalize on when using MINI-EXPLOR are the built in random number generator and the 'expand' program which was written by David Carr, a computer science major, at Syracuse University. First, the random number generator activated by the MINI-EXPLOR function NE (Min., Max.) used in a program can lead to interesting designs not necessarily predictable in advance. Here the art comes in by making the right choice of what works and what doesn't work in the final image. Chance has often played an important role in artistic creation and can be used as a valid parameter in creating a work of art.

Second, the 'expand' program enables the student to enlarge his or her line-printer image to any proportional size by joining the side edges of the standard line-printer paper together for the increased width. The only limiting factor is how much paper is available. This property adds an exciting new dimension to the MINI-EXPLOR system and to computer graphic art in general. One advantage of our MINI-EXPLOR system is that for a non-real time system, it is fast on returning images. This is very important in teaching traditionally trained art students who aren't used to waiting around to see what they have created. Using TTYOUT, the DEC-writer terminals return the hard copy image almost immediately so the art student can make the necessary visual critique of the work and proceed to modify the results through further programming or by hand. Designing on a display is even faster, waiting on line for over an hour for a printout is very bad and quickly discourages the creator.

I have found that art students who enjoy writing programs tend to stay with MINI-EXPLOR and desire to delve deeper into how the total system works from a programming point of view. MINI-EXPLOR motivates art students to want to create their own language for their own personal visual statement and gives science students an opportunity to apply his or her previously acquired programming skills for a purely aesthetic purpose. The science and engineering student is able to create visual images, using a familiar medium which relates directly to his educational and personal experience, without spending long amounts of time trying to master a traditional art skill which requires the traditional art talent. By this method the aesthetic experience has a direct transfer to the student's daily technological involvement.

MINI-EXPLOR is also an excellent software system to teach FORTRAN programming graphically to anyone.

PSPLAT is a FORTRAN program written by Richard H. Blocher of Washington University, St. Louis, Missouri, and more or less simulates another program, SPLAT, which was written for artists by John E. Skelton and Daniel J. Donohue at the University of Denver. SPLAT stands for Simplified **Pro**gramming **L**anguage for **A**r**t**ists.

PSPLAT (poor man's SPLAT), is a simple non-numeric computer language which is an excellent software package for introducing art students to computer graphics and should be used as an introductory course. The language uses statements composed of words which the artist can understand in terms of non-technical language. Everything is categorized by commands under headings like Creation, Manipulation, Looping, Branching, etc. which eliminates the need for the art student to go through tedious programming details. The output is principally by cal-comp plotter but we have output on our display as well, and this system has animation capabilities. We run PSPLAT by batch on the IBM 370 computer and MINI-EXPLOR on the time-sharing DEC-10 computer. This gives the art student the experience of using both types of systems. With PSPLAT the art student is working with a pre-defined symbol, such as a circle, square, polygon or line and draws and manipulates these symbols by writing a program, which is closer in concept to the way a traditional drawing is made. The art student can often visualize his or her final design in his or her 'mind's eye' making the process very comfortable for the student and a friendly rapport is established quite rapidly between the person and the machine.

In many cases the designs generated with PSPLAT are extremely complex and would have taken much more time if they had been executed by hand. The creator is thereby freed from doing the work and has more time to devote to thinking about developing new concepts for design possibilities. What often results is that the final images would not have been possible using traditional media.

I call PSPLAT a closed system because it doesn't lend itself easily to change by adding additional subroutines. MINI-EXPLOR, on the other hand, is an open system and encourages the addition of subroutines.

What is remarkable about using PSPLAT is that art students have been able to develop individual styles fairly quickly and neither the software nor the hardware has greatly hindered this stylistic development.

As I mentioned before, art students who turn on to programming in FORTRAN and get aesthetic satisfaction out of the programming challenge continue working with MINI-EXPLOR. Students who prefer a more traditional output continue with PSPLAT, and some art students choose to work with the system which can best provide the desired results for their aesthetic concepts.

The limitations of the software and hardware have not greatly hindered the development of individual style and the results can or cannot have a computer-made look, depending on what the artist wants. All media and materials impose certain limitations on the artist in the same way that the computer does. Whether the visual images generated by a computer system will ever constitute a major art movement remains to be seen. But when the artist transcends his media and materials with his unique personal vision and strikes human chords of sensuality, existence and human essence, we get art. I believe a number of my students have created art with the computer and this strengthens my belief that the computer is the most important technological achievement that has been made for the arts and will affect the course of art greatly in the future.

There is still much to be done in terms of hardware and software for the arts. Computer systems have to be designed to respond with greater sensitivity to the artist's sensitivity and expectations. I believe, at this time, that the major interest in computer graphic science is in the area of moving graphics for film and video, but static art is still the major movement in the art world and computer graphic science can contribute greatly to the future of static art images.

Artists are always searching for new forms and new ways to express their visual concepts. To use the familiar pleases the masses and saddens the individual. I believe the computer has the potential to provide new ways for the artist to create new forms, and I am looking forward to the computer becoming a standard studio tool for the creators of the visual image.

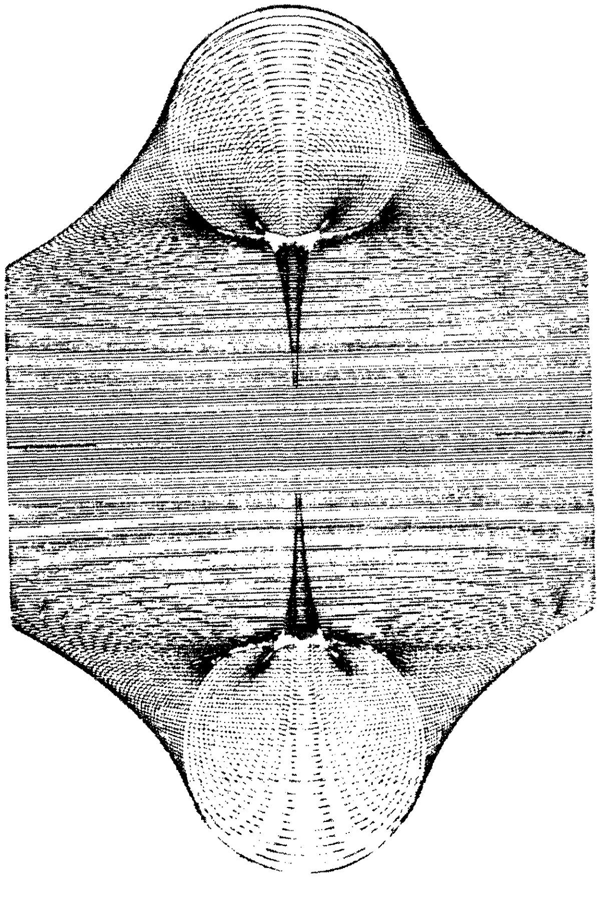


Figure 1—David Cox, "Pouring In and Out," 14" x 20", photo-etched intaglio, PSPLAT, IBM-370, Cal-Comp Plotter.

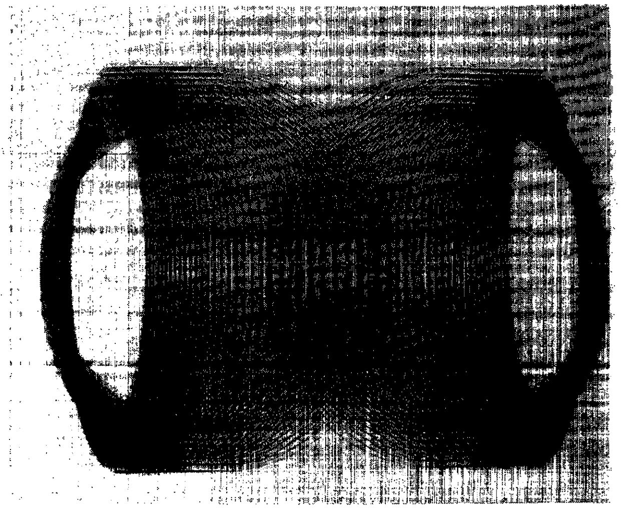


Figure 2—Ella Mears, "Eyes," 10" x 10", PSPLAT, IBM-370, Cal-Comp Plotter.

REFERENCES

Blocher, Richard H., *PSPLAT*, Washington University, St. Louis, Missouri.

Donohue, Daniel J. and John E. Skelton, *SPLAT—A Computer Language for Artists*, University of Denver, Denver, Colorado.

Knowlton, Ken, *MINI-EXPLOR—A FORTRAN-Coded Version of the EXPLOR Language for Minicomputers*, Bell Laboratories, Murray Hill, New Jersey.

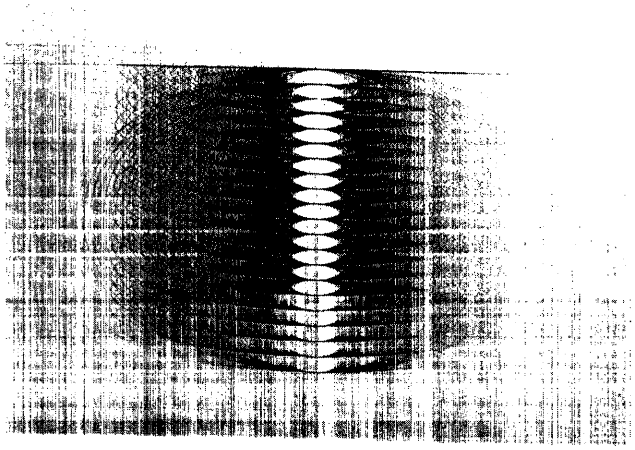


Figure 3—David Cox, "Great Multiple Circles," 10" x 18", PSPLAT, IBM-370, Cal-Comp Plotter.

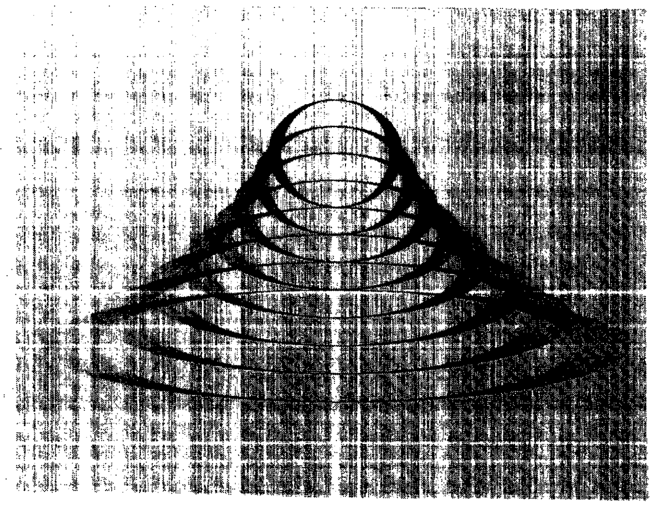


Figure 4—David Cox, "Cosmo," 10" x 12", PSPLAT, IBM-370, Cal-Comp Plotter.

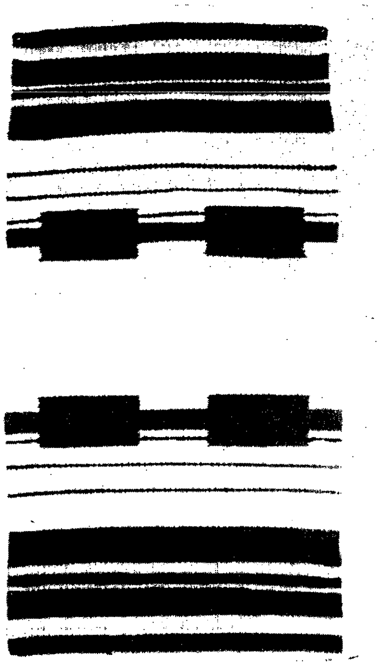


Figure 5—Susan Sirkus, "Woven Tapestry," 20" x 36", DEC-10, After EXPLOR lineprinter graphics, five colors, wool.

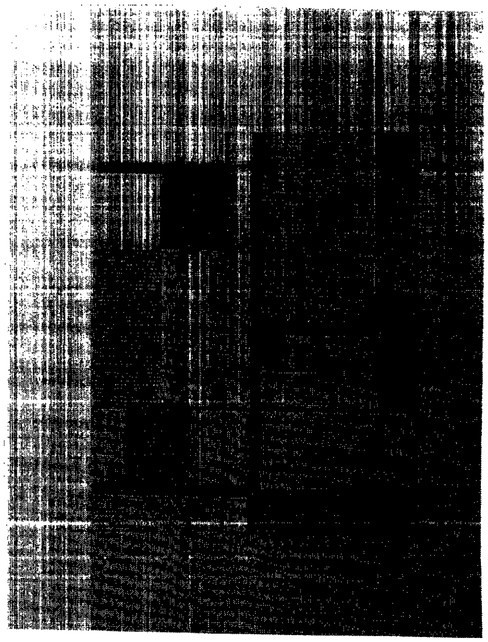


Figure 6—Don Leich, "After Mondrian," 14" x 18", MINI-EXPLOR, DEC-10, lineprinter.

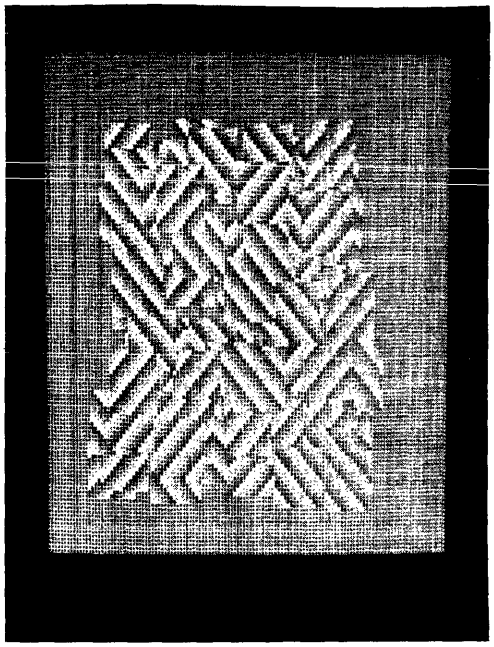


Figure 7—Don Leich, "Where Am I Going—Where Have I Been," 14" x 18", MINI-EXPLOR, DEC-10, lineprinter.

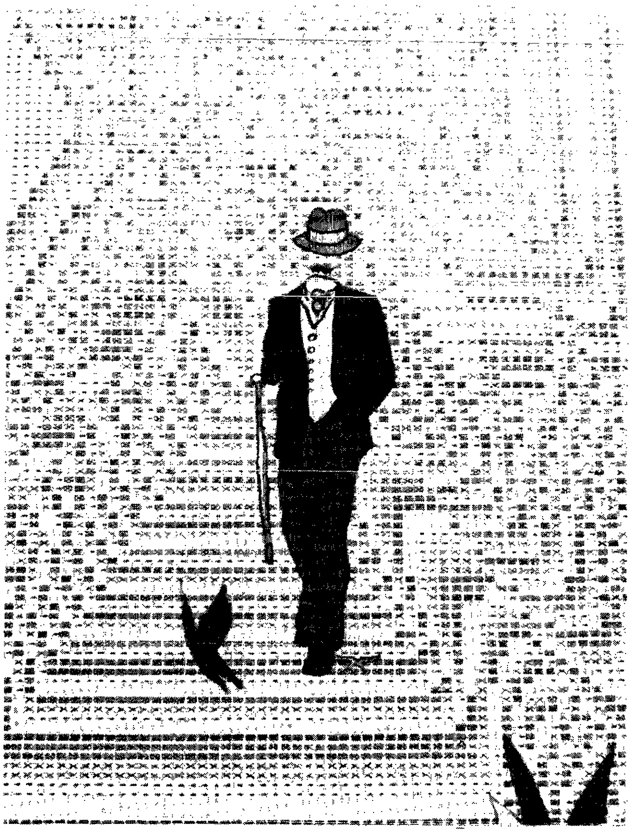


Figure 9—Bruce Maccurdy and Joseph Scala, "Strutting Through Computer Space," 8" x 10", mixed media, EXPLOR, DEC-10.

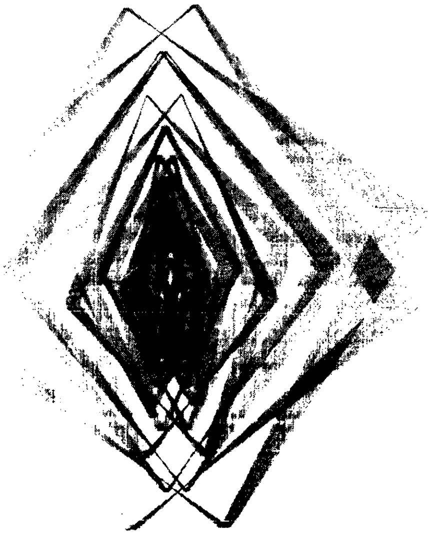


Figure 8—Ella Mears, "Crazies," 10" x 10", PSPLAT, IBM-370, Cal-Comp Plotter.

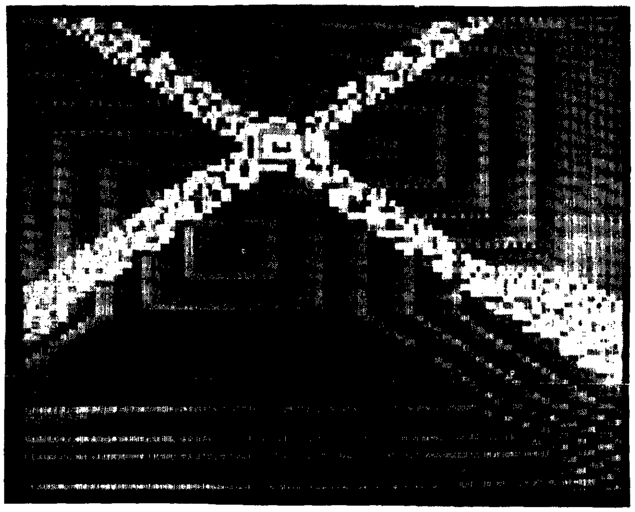


Figure 10—Joseph Scala, "Exploring II," 40½" x 52¼", EXPLOR, DEC-10, lineprinter, acrylic paints.

Artists and computers

by PATSY SCALA
State University of New York
Oswego, New York

ABSTRACT

This paper deals with the use of computers by artists as a means of aesthetic expression, considers several theories of the validity of art created through the assistance of computers, and discusses the author's personal reasons for beginning to use the computer as an artistic tool.

Sometimes, in the eyes of the uninitiated public, there is a vast dichotomy between artists and persons who use machinery, particularly electronic machinery such as the computer. Artists, according to public opinion, are intuitive, somewhat irrational, almost always illogical, often slightly eccentric, and create by some internal inspirational force, some muse, which takes the artist by the hand and leads him or her to the ecstasy of the act of creation.

Scientists, and especially those scientists who work with computers, are often considered to be highly rational, totally logical, definitely non-eccentric. Computer scientists create through a completely intellectual comprehension of the tools with which they work, with inspiration a folly to be tossed at the artist.

Contrary to popular opinion, and quite fortunately, none of this is true. At least, none of it is *entirely* true. Sometimes artists create with a high degree of logic and intellectual rationality. Often the work done by computer scientists, even in the writing of new programming languages, borders on conceptual art.

In reality, there are few differences between artists who use computers and computer scientists who use computers, except in the intent of the user, and in the knowledge of the software and hardware being used. Computer scientists use the computer to solve previously insoluble problems, to do mathematical modeling, to illustrate phenomena which could not be illustrated in any other way—for a multitude of scientifically necessary reasons.

But why do artists use computers?

Throughout the history of art, the artist has sought new means for expressing ideas. Many of the ideas expressed through art, and the tools used in the expression of these ideas, have been direct reflections of the existing ideas and tools in the society in which the artist lived. Take Michelangelo for example. If he were living in this country today, he would probably have painted "The Creation" on the walls of the White House instead of on the ceiling of the Sistine Chapel. Only it wouldn't have been "The Creation." It would have had a Bi-Centennial theme.

Paint itself, long taken for granted as a tool for artistic expression, was at one time many centuries ago, a new technology.

So the artist adapts.

Today we live in a highly technological society. We have quietly gone out of the Age of Machinery and discovered ourselves plugged into the Age of Electronics. Everywhere we look, we are surrounded by the ramifications of the two most pervasive electronic devices ever created: television and the computer. The influence of television is more apparent to persons in the general public; approximately ninety-seven percent of the homes in the United States have at least one television set. In fact, our country boasts of more television sets than toilets. It has become, for the masses, an accepted medium, and a very visible one. For this reason, it is non-threatening to a person in the general public, although, for the intellectual elite, it is often a very real problematic threat. The entrance of the computer into the lives of Americans has been more subtle. To the uninitiated, the computer is a techno-monster which messes up bank statements, puts erroneous amounts of money onto paychecks, and generally terrifies the mass public, who see it as a machine which could ultimately overthrow humanity.

It is, of course, none of these. The computer, programmed by persons who know how to do so without error, is one of the most important tools of learning that has ever been invented. It can, essentially do anything the intelligent programmer wants it to do.

An artist living in this highly technological, electronic world has basically two choices as to how he or she will handle aesthetic expression. The artist can either choose to ignore these new electronic tools, and deal only with traditional tools of art, thus making the statement that electronic means of artistic expression are invalid and anti-art. Or the artist can choose to use these tools in an attempt to create thus-far totally new imagery, or in an effort to prove that electronic technology can indeed be humanized.

It is my feeling that the artists who are currently tackling the new technologies to create art are making the most significant contribution to the world of art, despite the fact that most art critics disagree with this theory.

Poet T. S. Eliot, in an essay entitled "Tradition and Individual Talent," offered a definition of art which stated that a work could be considered art if it, first of all, fits into some prior tradition of art, and secondly, if it adds something new to this tradition which significantly changes the tradition of art.

Artists who ignore the computer and television are fitting into the established tradition of art; through ignoring the technologies with which we live, however, they are adding precious little to this tradition.

Artists who are using the computer to create aesthetic statements are obviously adding something new to the tradition of art, not only through the types of imagery they are able to achieve, but also through the use of a totally new tool. And contrary to the arguments of many artistic traditionalists, computer artists do fit into the general tradition of art. Static computer graphics have a parent in painting; still graphics can be judged aesthetically by the same standards that static art has been evaluated through the centuries. Animated computer art also has a place in the general tradition of art. First of all, good computer animated art, whether it is implemented in film or video, also fits into the tradition of painting, since each single image in a computer animated piece is carefully composed, utilizing juxtapositions of color, form, perspective and planar relationships. Computer-animated art also fits into the tradition of film, and, through its often mathematically choreographed imagery, into the tradition of the fluid movement of music, albeit visualized music.

The field of computer art is still too new for a solid aesthetic to have been formed, stipulating which computer-assisted art is truly art and which is merely interesting experimentation. Some basic ground rules can, however, be stated. First of all, the intent in the creation of the work can be considered. If a person manipulating computer imagery intends to create art, his or her work must at least be considered for artistic judgment. If, in addition to this, the finished computer-assisted art presents to the viewer imagery which is new, unconventional and visually interesting, it deserves further consideration as art.

Also, in the art world, it is common knowledge that if other artists accept a piece of art as aesthetically valid, it takes still another leap toward being seriously considered art. If, therefore, artists relate to a piece of computer-assisted art, it has gone one more step toward qualifying as art.

Problems do arise, however, in evaluating computer-assisted art. Although some artists using the computer have a strong understanding of the internal operation of computer hardware and software, there are many more who do not and who are, therefore, compelled to create their art in collaboration with a computer scientist or programmer. This in no way invalidates the value of the art, although, in the communication process between artist and computer scientist, many difficulties in language must be overcome. For example, an artist might say to a computer scientist, "I'd like a piece of film which bursts forth from an epicenter, then moves around in circles for a few minutes, then swirls across the screen in something that looks like a tornado." How does the computer scientist translate this into a viable programming assignment? With difficulty. But it can be done, and often the results are far more striking than if the artist had manipulated the imagery alone, since the scientist can add his or her own knowledge of what can be done with the computer, thus adding dimensions that the artist might never have conceived alone.

Another parameter to consider in discussing computer art is the fact that some of the most aesthetically beautiful computer-generated imagery has been produced by scientists as an offshoot of their scientific inquiry. Since the intent in the creation of this imagery was not to create art, it probably cannot be considered art. But increasingly, as computer scientists see the beauty, as well as the scientific validity, of the output they produce, these scientists are making choices that certain of their images are indeed artistically valid, and therein enters the intent to show art.

What seems to be happening is that artists using computers are becoming more knowledgeable in the field of computer science, and computer scientists are becoming more aesthetically aware. Soon there may be a point at which computer scientists are computer artists, and computer artists become competent programmers.

My personal involvement in computer-assisted art arose as an offshoot of my work in video. I discovered that, using video alone, I was unable to obtain the types of imagery I visualized. The computer gave me a new means of creating visual input for my video tapes. I personally have chosen not to use straight computer-animated input for my video tapes; in order to produce the imagery I envision, I utilize computer input on film, then do video-graphic manipulations on this imagery. Some of my finished pieces retain the quality of the original computer-animated input; others have been so manipulated through video techniques that they no longer retain the look of computer graphic animation.

Whatever techniques artists choose to use in their manipulations of computer graphic art, it seems that the computer has become one of the most useful tools available today to the artist who is in tune with his or her times.

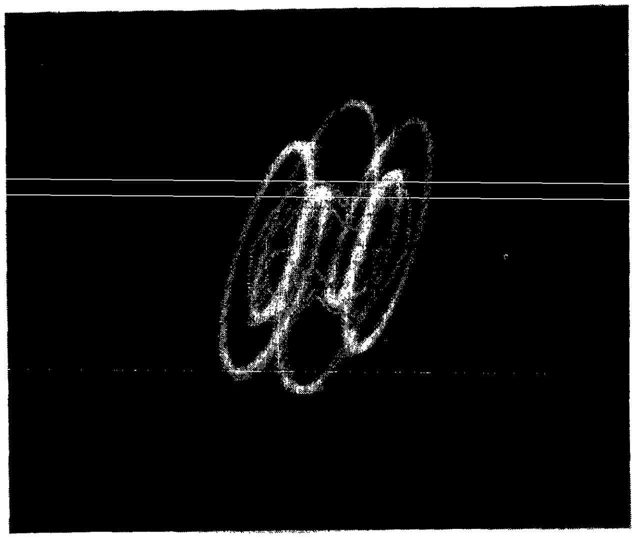


Illustration I—From SCOPE I, videotape by Patsy Scala. In this piece, fewer videographic manipulations were introduced in converting the analog computer imagery to video. The work retains much of the original quality of the original computer animation.

Illustration III—From WIPEPOEM, videotape by Patsy Scala. In this piece, red, green and blue refracted laser light was distorted by analog computer voltage and videographic manipulations. The piece retains little of the look of computer animation; yet it can be judged compositionally by the same standards that traditional paintings are judged.

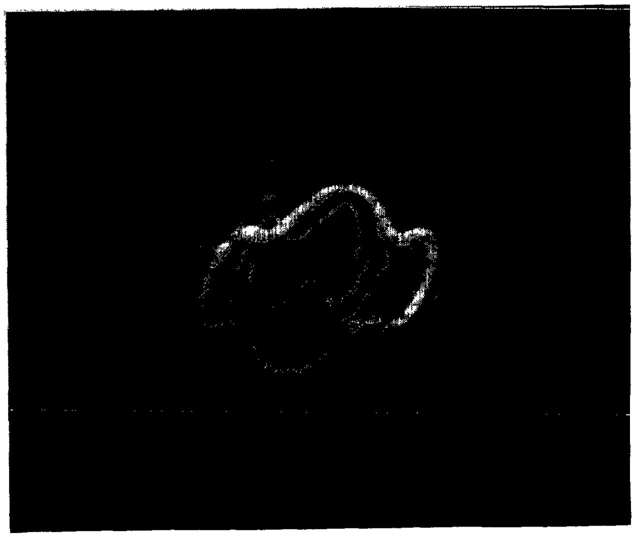


Illustration II—From SCOPE II, videotape by Patsy Scala. In this piece, composed of analog computer generated imagery, the videographic manipulations do not obliterate the "computer look" of the piece.

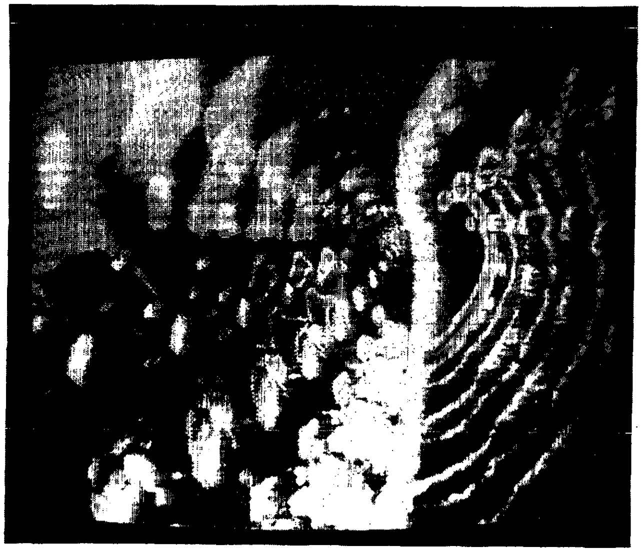


Illustration IV—From WIPEPOEM, videotape by Patsy Scala. Again, the analog distortion of laser light, and three levels of videographic manipulations leave the finished piece with little of its original computerized quality.

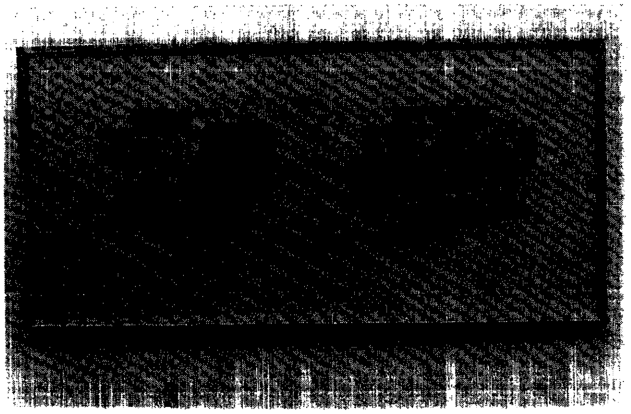


Illustration V—AZTEC I, by Joseph Scala. This piece, generated through Kenneth Knowlton's EXPLOR system, was hand colored by the artist to give it the look of Aztec tapestry.

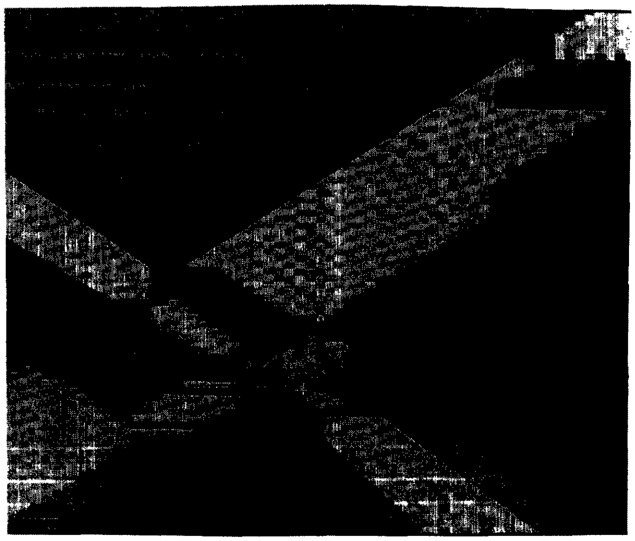


Illustration VI—EXPLORING III, by Joseph Scala. This large painting, done on computer printout paper, was based on Kenneth Knowlton's EXPLOR program, yet the addition of paint gives it a very different character from typical computer art.

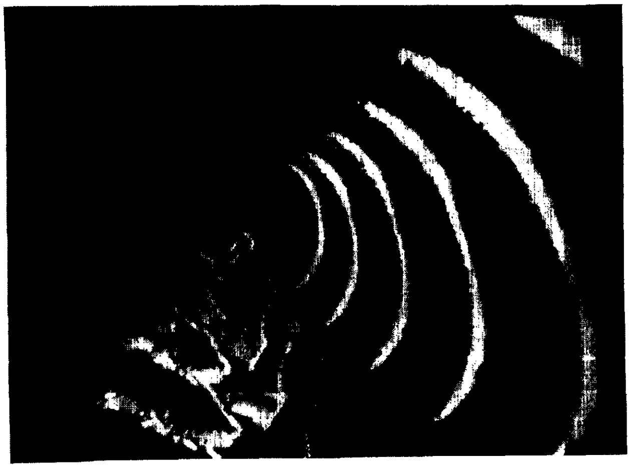


Illustration VII—From WIPEPOEM, videotape by Patsy Scala.

The digital component of the circle graphics habitat

by THOMAS A. DEFANTI
University of Illinois at Chicago Circle
Chicago, Illinois

ABSTRACT

This real-time interactive computer graphics system derives from the author's dissertation[1] at the Ohio State University (National Science Foundation Grant GJ-204, Charles A. Csuri, project director). The system, called "The Graphics Symbiosis System" or "Grass" was first designed to help artists interactively explore computer art without the constant companionship of a programmer. Over the past three years, it has been expanded at the University of Illinois at Chicago Circle (Figure 1) and is now the image generation portion of a short-order full-color animated videotape production facility called "The Circle Graphics Habitat." Combined with Dan Sandin's Image Processor, the system[2,3] is sufficiently powerful and flexible to be used in real-time performance context[4] here at UICC.

INTRODUCTION

The hardware is a standard PDP-11/45 computer with a Vector General 3DR display scope. In addition, we have a data tablet, thirty channels of analog input devices (dials, slide potentiometers, joysticks, etc.) and several channels of analog output (built by Larry Leske) to drive the Image Processor.

Figure 1—The Grass user's console

Primary inspiration for the structure of the language was the DECsystem-10 operating system and its text editor TECO, and Ron Baecker's animation system "GENESYS."[5] About ten person-years of programming effort up to now has resulted in about ten thousand lines of code in assembler. Persons primarily responsible for implementing the software will be identified by their initials as we progress: Tom Chomicz (TC), Dean Daniele (DD), Tom DeFanti (TD), Mike Dearing (MD), Nola Donato (ND), Manfred Knemeyer (MK), Gerry Moersdorf (GM), T. J. O'Donnell (TO), Ralph Orlick (RO) and Bob Rocchetti (BR). Gerry Moersdorf was responsible for about half the code in the Ohio State implementation. Grass currently has six running installations.

This paper will communicate the program structure of Grass. It would be good, however, to first describe briefly what is expected of the system by university authorities and users. The system, with the Image Processor, is used by professors, instructors and advanced students in preparing animated educational materials in less than geological time. Educational animation makes demands on a system which are computationally and conceptually more complex than standard plotting. The system has to be both easily learned and powerful enough for use by expert programmers in animations like continental drift or a dynamic explanation of how television works. A friendly review of the Circle Graphics Habitat may be found in Reference 6.

Educational animation systems are exciting to develop but not always as much fun to use. Educational animation is rarely free and loose. Fortunately, to continue the good feeling of pure, uncompromised creation, we now have an annual event in April which, like a faculty recital in music, is a performance, but of animated 3-D images and color image processing along with live music (Figures 2 and 3). We believe that jamming on equipment like this demands more from the human engineering side of design than ordinary real-time graphics. More on the philosophy of

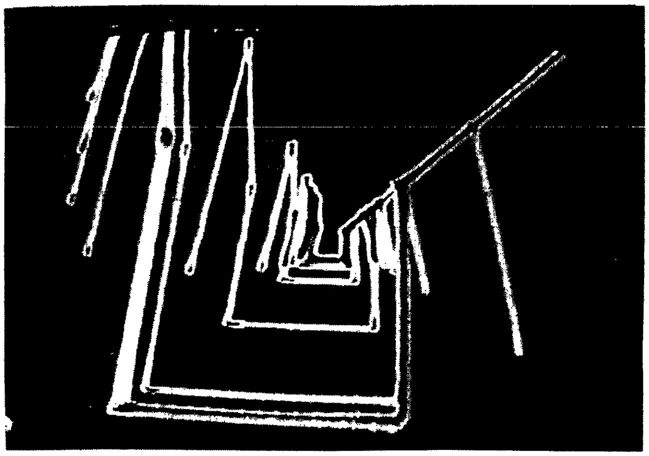


Figure 2—A still from 'Peano Boogie' by Sandin, Morton and DeFanti, a live performance. Colors are sky blue, raspberry red, lemon yellow and orange orange

the Circle Graphics Habitat may be found in Reference 2.

For the technical discussion that follows, it is assumed that the reader has knowledge of interactive graphics to the level found in Newman and Sproull.[7]

TECHNICAL DESCRIPTION OF THE GRASS LANGUAGE

Grass has two essential types of primitives: commands and pictures. Commands are kept as ASCII strings terminated by carriage-returns. Pictures (often called "sub-pictures" in the literature) are user-defined displayable lists of 3-D absolute vectors compiled into binary code acceptable by the Vector General display direct memory access processor. On disk, both commands and pictures are normally kept as ASCII strings (we always wanted to be able to read what was on the disk). Both commands and pictures in Grass are higher-level primitives, and this concept is essential to the design of the system. In addition, both may be broken down into the lowest level components if desired (i.e., ASCII characters and x, y and z endpoints) and both may be grouped into hierarchies whose elements have much the same behavior as the primitives themselves. The user first learns the command primitives like ROTATE, MOVE and SCALE to manipulate graphical images. Non-programmers do not have to think of images as endpoints or manipulate instructions as individual characters. This differs considerably from the approach taken in typical FORTRAN graphics packages where the user is assumed to be an expert programmer.

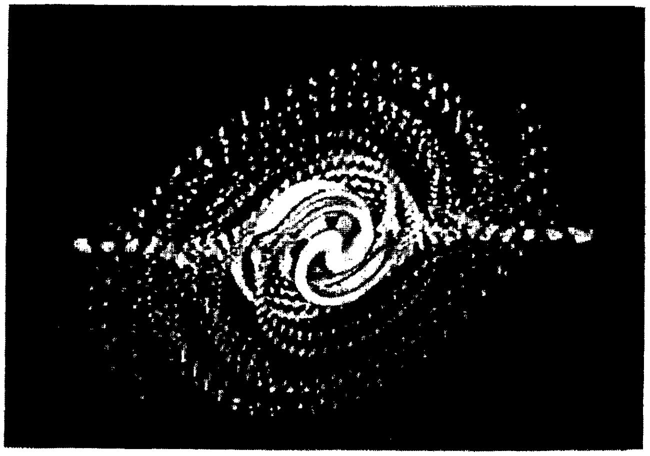


Figure 3—A still from 'Wednesday Night Spiral' by Sandin and DeFanti, a live performance

USE AND CONTROL OF PICTURES

Pictures in Grass have user-assigned names and are displayable lists of vectors prefixed in core by a control block. The user indirectly communicates with the control block for each picture by using commands like ROTATE, MOVE and SCALE rather than by setting bits. At interrupt level, the system updates and writes these control blocks out to the display's internal registers which drive the hardware features of the display (see Reference 8 for a complete description of the Vector General Scope). Updating of the control blocks is done at programmed interrupt level six and the vector display is done by direct memory access over the UNIBUS. The maximum number of pictures concurrently displayed is an assembly feature, presently set to sixty. To the user, all pictures are effectively processed in parallel. The commands that set up to use the hardware are:

MOVE PIXNAME,x-DEVICE,y-DEVICE, z-DEVICE

> Translates picture named "PIXNAME" to current values of the x-, y-, and z-DEVICEs until disabled (that is, the command has to be issued one time only, but any time the DEVICEs are changed, the update is done automatically). A DEVICE is defined as either an analog input device or an integer variable. Example: MOVE TITLE,D2, D3,D4 moves the picture "TITLE" on dials 2, 3 and 4.

SCALE PIXNAME,DEVICE

> Scales all three dimensions of the picture on the DEVICE until disabled. Also available are SCALE/X, SCALE/Y and SCALE/Z which scale in individual axes. Example: SCALE/X CIRCLE,A scales the circle horizontally on variable A.

SETINT PIXNAME,DEVICE

Sets the intensity of the picture to be continuously variable on the DEVICE until disabled.

SETCUT PIXNAME,DEVICE

Sets the z-axis depth cueing and z-axis cutoff feature of the scope until disabled.

The system hardware excels at rotation. Consequently, the ROTATE command, a very high-level primitive, has many options:

ROTATE PIX,AXIS,SPEED-DEVICE

Gives simple rotation about the x, y or z axis (specified by AXIS) at a constant speed of rotation determined by SPEED-DEVICE. If angular position rather than speed is desired, ROTATE/D is used for this and all the following ROTATE commands. (By MK and DD.) Example: ROTATE TETRA,Y,K rotates "TETRA" around the y-axis using variable K for the speed of rotation.

ROTATE PIX,AXIS,SPEED-DEVICE,
TILT-DEVICE

This rotation uses the second DEVICE to control the angular position of the axis of rotation in the plane through the origin perpendicular to the AXIS specified. It is a strange rotation to describe on paper but it is a highly useful rotation for interactive use. (By MK and DD.)

ROTATE PIX,AXIS,SPEED-DEVICE,
TILT-DEVICE,X-DEV,Y-DEV,Z-DEV

This rotation gives arbitrary origin rotation capability. It tends to produce elliptical rotations, the backbone of complex animated sequences (as in Disney's Fantasia). Again, it is non-intuitive and requires feedback to use. (By MK and DD.) Example: ROTATE COPTER,X,D0,D1,D2,D3,D4

ROTATE PIX,7,SPEED-DEVICE,X1-DEV,
Y1-DEV,Z1-DEV,X2-DEV,Y2-DEV,Z2-DEV

The "seven-dial" rotate allows the user to specify the endpoints of an arbitrary axis of rotation. It is the rotation preferred by programmers describing scientific phenomena in terms of rotation. (By TC.)

ROTATE/X PIX,DEVICE
ROTATE/Y PIX,DEVICE
ROTATE/Z PIX,DEVICE

These rotations are compounded with any of the above rotate commands to produce more complex effects. Grouping of pictures allows further compounding of transformations.

PATHMOV PIX,PATH-NAME,SPEED-DEVICE

This command tangentially moves the PIX along a PATH (which is simply any picture, displayed or not) with the given speed. It is basically an extension of Baecker's p-curve.[5] (By TC.)

These commands stay in force and cause constant updating to the values of the DEVICEs until disabled by the FIX or RESET commands. None of the above commands actually change the vector list since all the functions are done by the hardware.

Many commands do, however, change vector endpoints. The most used ones are:

SMOOTH—does a binomial smoothing of a vector list (BR).
PERSP—does perspective projection (TC).
CLIP—clips one picture against another (like film matting or video keying) (TC).
WINDOW—does normal 3-D windowing (TC).
SHADE—shades in outline with vectors (BR).
SOFT—carries out the hardware transformations on the vectors by software (DD).

The precise syntax of these commands is contained in the on-line HELP file, a copy of which may be had on request.

As is usual with sophisticated refresh graphics systems, the pictures appear to be parallel processed. They are controlled in parallel by turning dials (which are polled at interrupt level 30 times a second) or by manipulating variables. The user can also get at the individual endpoints of pictures with the GETPOINT and ZAPPOINT commands, build pictures with the PUTPOINT command, or draw them in using the tablet or other digitizers. (There is also complete software for text appearing on the Vector General (by BR and TO)).

In addition, the user can group pictures together and create a tree structure hierarchy of control on these grouped pictures. Pictures are grouped for convenient reference, and the groups respond to all the hardware transformation commands just as pictures do. Most often, the GROUP command is used to create multiply articulated structures like Professor Csuri's airplanes, helicopters and witches on propeller-driven broomsticks[9] as well as very complex rotations and translations. Grouping may be carried on to 59 levels (the same assembly feature as before). Note that the user does not have to know about tree structures, lists and pointers to use any of these commands because the system housekeeping does all the chaining. For knowledgeable users, the TREE command gives a schematic of the hierarchy developed. A two hour-long videotaped lecture describes the internal workings of the system algorithms.[10]

Storage allocation (by TD and ND) is by a "best-fit" algorithm. About 10K of 16-bit words is available to the user for vectors, text, command strings and disk-resident command modules. Many Grass commands, especially user aids like TREE and software transformations like PERSP, SHADE and CLIP are not core-resident. Code (by GM and RO) automatically fetches the appropriate modules into core and executes them. They are automatically deleted. When done, the total overhead amounts to about a tenth of a second.

Garbage collection (by TD and ND) dynamically tries to maintain large blocks of storage. It is called by the user commanl "DELETE" or automatically invoked by system housekeeping whenever appropriate.

As an example, a videotape[11] to illustrate that 3-D rotations are not commutative (Figures 4, 5 and 6) was done using the following code:

```
GETDSK BFLY1            (get butterfly from disk)
COPY BFLY1,BFLY2        (make a copy)
ROTATE/D BFLY1,X,D1     (rotate on x axis)
ROTATE/Y BFLY1,D2       (then compound with
                          y rotation)
SETINT BFLY1,D3         (control intensity on
                          dial 3)
ROTATE/D BFLY2,D5       (rotate on y axis)
ROTATE/X BFLY2,D6       (then compound with
                          x rotation)
SETINT BFLY2,D7         (control intensity on
                          dial 7)
GETDSK 3DAXES           (get the axes up)
GROUP 3DAXES,           (group and call the
  BFLY1,SAM               group "SAM" BFLY2
                          gets in for free)
ROTATE/D SAM,X,S1,S2    (rotate the whole thing)
```

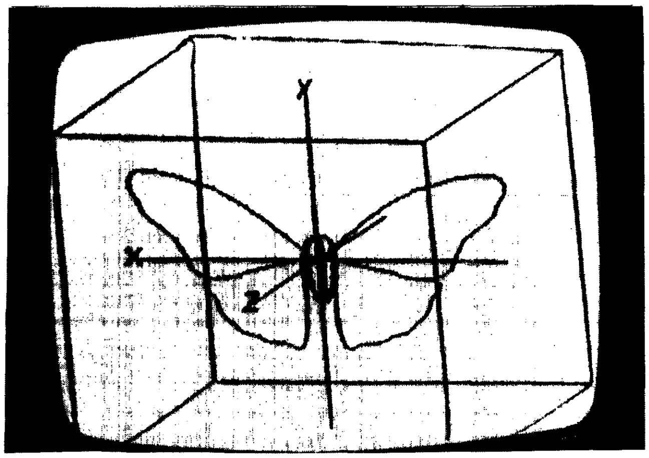


Figure 4—The butterfly in its original position

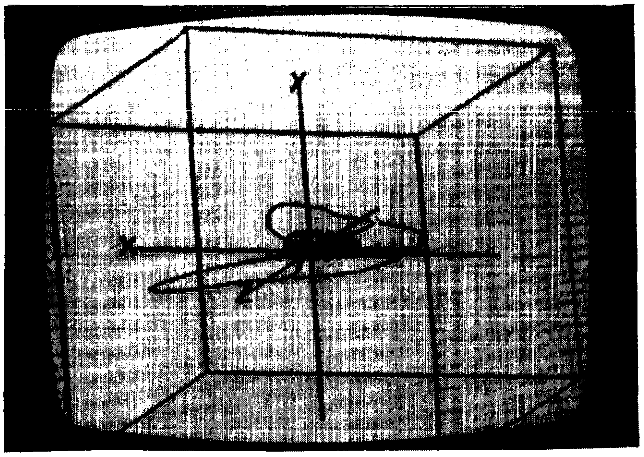


Figure 5—The same butterfly rotated ninety degrees around the x-axis and then ninety degrees around the y-axis

Now, by turning dials 1 and 2 enough to get ninety degree rotations about the x then y axis, the first butterfly takes one position. Turning dials 5 and 6 enough to get similar rotations about y then x axes, the second butterfly goes to a different position. It is quite easily seen that the two are not equivalent. That is, 3-D rotations are not commutative. (Rotating the group "SAM" allows the third dimension to be seen more clearly. The intensity controls allow independent fading of the two butterflies for clarity.) Given the existence of the butterfly and the axes on disk, the entire videotaped sequence took less time to produce (about fifteen minutes—some of which was dedicated to aesthetic judgment and color choice) than to describe.

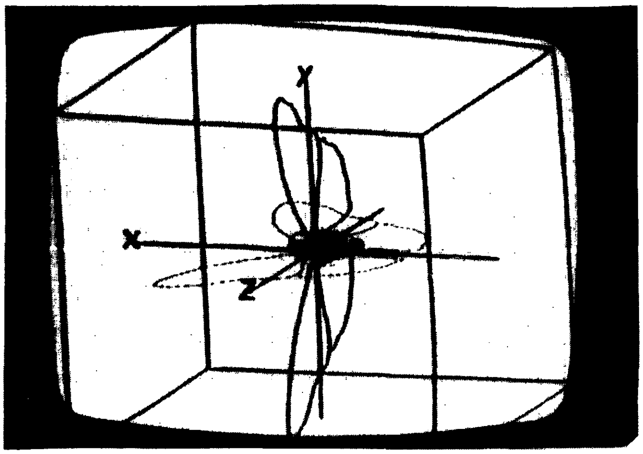


Figure 6—The same butterfly at half intensity contrasted with another butterfly rotated ninety degrees around the y-axis and then ninety degrees around the x-axis

USE AND CONTROL OF COMMANDS

The foremost design criterion of the command language in Grass has always been habitability, a term adopted[12] which means the quality of a system that makes it easy to learn and use. In many ways, the linguistics of graphics languages are quite unnatural for describing animation and many people do much better by waving their hands. On the other hand, the power of linguistic structures is undeniable, especially when modeling scientific data.

Programming on someone else's system is always frustrating. What really matters is whether you get anything done while being frustrated. We have tried to design a system to help users at all levels to get things done.

The basic tenets of habitability are usually obvious—mnemonic command names, predictible syntax, good error messages, high feedback—but they are rarely implemented in full because the detail work in coding is immense. The leisure time for attention to such detail is hard to come by, especially if the users are overwhelmingly power-conscious and impatient programmers. In a short-order videotape laboratory where users are by and large professionals donating their time to improve the quality of their classes, non-alienation is the item of highest priority. General fun and productivity seem to follow. Again, more of this philosophy is found in Reference 2.

A command in Grass is a string of characters terminated by a carriage return (CRLF) or a semi-colon in the case of multiple commands on a line. In order for the system to process a command in any context, originating from anywhere, the string of characters is simply passed to the line processor (LINEP) which interprets the string, dispatching to the proper command module. There are two general formats:

```
COMMANDNAME ARG1,ARG2,ARG3,...
  (for most commands)
    Example: SCALE WIDGET,D0
VARIABLE=EXPRESSION
  (for FORTRAN-style commands)
    Examples:
      A=A+10
      K=K-D0/2   (note use of dial 0)
      FA=ATN(FP)+SQR(FQ*(FD-FE))
```

If the command is not core-resident, it is fetched from the disk (all disk-resident commands are written in position-independent code), executed and deleted.

Commands are typed in line by line on the video terminal (VT05). Since many of the commands set up processes at interrupt level, the system may be used exclusively on a line-by-line basis as with a text editor. The previous rotation example was done this way, as was a twenty-minute film with witches chasing butterflies and airplanes flying around the globe, at a time (1972) when the language had only line-by-line capabilities. Almost all programs written in the language start as a few commands typed in and tested one line at a time.

The next step in command processing is to take commands stored in a file so they in essence become like the roll of a player piano, or the paper tape for a milling machine. Many text editors allow this type of command usage. This grouping of commands is often called a "macro."

Macros in Grass are simply groups of commands—a concept easily grasped by all our users. In compiled languages, macros are called "subroutines," but this terminology was not chosen because Grass macros are often not subordinate to anything conceptually. Macros do not require preambles, declaration statements, end statements or other formalities associated with subroutines.

To give the player piano roll in Grass a variable fast-rewind and fast-forward, control is transferred with the SKIP command. Its argument (e.g., SKIP 3) specifies how many CRLF's to pass over forwards or backwards. The argument may also be a label, in which case the transfer is to that label local to the current macro. Transfer to other macros is with the DO command whose argument (e.g., DO SETUP) is a macro. If this macro is not core-resident, it is automatically fetched from disk and then interpreted. The CALL command is similar except that it uses the system area of the disk and tries to find a compiled version of the macro (see below for details on the compiler). As will be seen later, the syntactic form of the macro call with parameters is very close to the form of commands so that system macros can appear to be system commands to the user.

Conditional branching and command execution is done with the IF command whose syntax is simple:

```
IF VARIABLE=EXPRESSION,ANY COMMAND
  Examples:
    IF A=B,SKIP -5
    IF D0 GT 0, IF A LT -100, DO FIXUP
```

In any case, control returns to the statement following the DO when the indicated macro is finished.

Macros may be generated in several ways. The system editor can be used to enter and change ASCII files on the disk or in core. Macros may also be created by typing a name followed by a colon as in the following example:

```
SETUP:<GETDSK GLOBE
SCALE GLOBE,D0
ROTATE GLOBE,X,TX,TY
  (using the tablet x and y)
GETDSK TITLE
MOVE TITLE,D6,D7,D8>
```

To execute this macro, one types "DO SETUP" or simply "SETUP" (providing it is not a system name). Often, immediate execution is desirable. The "unnamed" macro, entered thus:

```
<ANY COMMANDS
  . . .
  . . .
  . . .>
```

is automatically executed upon typing the final angle bracket. In addition, when this type of macro finishes, it is automatically deleted. Only named macros may be stored on the disk.

The system uses a VT05 video terminal at 2400 baud for user communication. This choice was originally made (by MK) to keep the Vector General screen free for images to be recorded by a camera. One version of Grass uses a keyboard interfaced through the Vector General and space at the bottom of the screen for the same purpose. At any rate, the system can afford to be fairly wordy with messages without much delay.

Being strings, macros can generate other macros using the string variables ($A through $Z in Grass) and the string manipulation primitives given by concatenation and the SEARCH command (MD and TD). This feature, exploited only by fairly experienced users, allows very plastic fabrication of executable statements in Grass. String variables are executed by putting them alone on a line after construction:

```
$A='GETDSK GLOBE
SCALE GLOBE,D0'
$A
```

Passing parameters is usually clumsy in programming languages. Indeed, we have only recently implemented a habitable way of passing parameters between macros (by RO). Like everything else in the system, macros are used interactively. Rather than burden users (who often cannot yet write macros) with having to know which parameters to supply, macros are usually written to ask questions:

```
SETUP:<PROMPT "WHICH PICTURE DO YOU
  WANT"
INPUT $A        (system types a "?" and waits
                user then types in a name)
GETDSK $A;SCALE $A,D0     (a multi-command
                          line)
ROTATE $A,Y,D1>
```

Effectively, any picture on the disk may be gotten, scaled and rotated with this macro. Similarly, one can input numeric values (e.g. INPUT FA) as numbers or expressions. The PROMPT command is the general typed output command, and may be used to print strings or numbers and combinations thereof:

```
PROMPT "THE SQUARE ROOT OF 1000 IS ",
  SQR(1000)
```

However, once the user is familiar with the macro, he may type "DO SETUP,GLOBE" or simply, "SETUP GLOBE" which looks like a Grass command. Either construction may be imbedded in another macro without any reprogramming of the original macro—an important feature since many users cannot decipher complex macros written by others. Note that the PROMPTs are automatically suppressed as long as enough arguments are supplied. If the user leaves off an argument, the system will wake up the PROMPTs and start asking the questions again. Of course, overrides are available to force INPUTs or PROMPTS if desired. Arguments may also be passed in global variables.

Variables in Grass have fixed names and are either local to macros (LA-LZ, fixed; EA-EZ, floating) or global (A-Z, VA-VZ, WA-WZ, fixed; FA-FZ, floating; $A-$Z, string; and AA-AZ, fixed and floating arrays). Analog inputs are global (D0-D9, dials; S0-S9, slide potentiometers; JX,JY,JZ and KX,KY,KZ, joysticks; P0-P3, more dials). Digital inputs include the tablet (TX,TY,TZ or pen-press) and function switches (FS0-FS15). Analog outputs are global too (OA-OH). Variables are prenamed in Grass because otherwise the interpretive overhead for arithmetic would really be immense. Prenaming also eliminates the need for declarations of variables (except for array dimensions).

The external inputs are polled every 1/30 second by the system. Currently under construction is a flexible input box with variety of connectors and amplifiers to aid in prototyping new input devices such as your body or musical instrument to produce a set of Grass variables. Having so many analog input devices, by the way, may seem confusing to the reader, but these physical extensions to the system can be intuitive and therefore easy to use.

Macros work despite the parsing overhead because the primitives of the language are generally rather high level and parsing is only a fraction of the code executed in doing the command. The higher the level of the primitives, the more practical the interpreter becomes. Note also that often one does not care how long something takes to parse as long as it is done in say, less than 1/60 second.

However, for low level primitives like addition or expression evaluation, the interpreter may execute a thousand instructions to add one to a variable. This is a major reason compilers are still preferred for arithmetic calculations. As soon as we started doing scientific animations as well as computer art, a fast arithmetic capability became essential. Thus, The Habitable Compiler was written (by RO). It takes assignment statements and some commands (notably GETPOINT, ZAPPOINT and PUTPOINT) and compiles them into PDP-11 machine code which executes very quickly. Whatever the compiler does not understand, that is most commands, it keeps as ASCII strings which are passed to the resident interpreter during execution. Thus the compiler retains the benefit of the interpreter and yet gains the speed of compiled code where essential. One usually debugs the macro first and then, if speed is a problem, compiles it. The compiler is disk-resident and rarely takes more than a second to load in and do its job. Compiled macros may be stored on the disk (in binary) and recalled at any time. In addition, macros that contain only arithmetic code are re-entrant so they can be set up to execute at interrupt level with the VIP (Very Important Program) command. VIPed macros are used when variables must be calculated perfectly in synch with the display refresh or when a higher priority task is useful. Few higher-level languages allow users to schedule subroutines at different priority levels.

To give a quick idea of the compactness of Grass macros, Donald Warren Collins wrote an architectural preview system[12] first on the IBM S/360 in FORTRAN taking some 132K 32-bit words, then in PDP-11 RSTS-11 BASIC taking 28K 16-bit words with multiple overlaying. Finally, he rewrote it in Grass, thus making it interactive. The Grass version took about 2K 16-bit words of storage, not including the 13K interpreter, of course.

DRIVING MACROS

Most of the contribution this system has made to habitability in graphics is noticed when actually executing and debugging macros. The constant real-time user control combined with the analog input devices makes this system usable as a performing instrument. Some of this control comes from parallel processing of pictures and, if the user desires, macros as well.

Animation often involves several more-or-less independent things happening simultaneously. Grass can be asked to set up a ring structure of macros so they can be executed in parallel. With two macros, the commands are interleaved. More than two macros requires grabbing lines from each one, one at a time, round-robin fashion. In addition, unless specifically requested otherwise, a macro in this ring structure automatically starts over again when finished. All this housekeeping is initiated by the DOLOOP command (by TD):

```
DOLOOP MACNAM1,MACNAM2,...
```

where the MACNAMs may be macros or compiled macros. The unnamed macro is retained for convenience in setting up background jobs:

```
DOLOOP <A=A+D0/100
B=A*2>
```

and so on. The system continuously listens to the VT05 keyboard for a line to be typed interactively and slips it in, executing it with no noticeable delay in most cases. Provisions for one macro waiting for another to complete and selective removal from the ring also exist.

As might be imagined, there is some overhead associated with the DOLOOP feature, but compiling the macros usually helps make the overhead quite unnoticeable. The need for and desirability of a parallel execution structure in graphics is more fully developed in Reference 3. Alan Kay's SMALLTALK language for children also implements this type of parallelism for 2-dimensional graphics.[13]

Two major conceptual simplifications of using macros result from parallel execution. First, the user can develop small animation sequences separately and then combine them later with little or no reprogramming. Without the parallelism, a total rewrite of the macros would be required. Second, the user may wish to incorporate macros written by others in his animation sequence, macros whose logic he may not understand. Again, without the DOLOOP structure, this would be very difficult.

Grass also has well-developed "panic-button" control structures (by TD) to abort or temporarily interrupt macros. First, CONTROL-C (holding the control key while typing a "C") stops any macro or compiled macro, kills the DOLOOP structure, stops any output, cleans up any scratch files like unnamed macros and sets the user back to command input level. It is the most common way to exit a macro in an infinite loop.

CONTROL-W temporarily stops printout until pressed again (2400 baud is too fast to read) and CONTROL-O cancels any output.

The real crowd-pleaser in the system, though, is CONTROL-S. It suspends execution of any macro or compiled macro, even if DOLOOPED. The user is put in command level and he can type commands to check variables, or anything else. The macro continues when the user types "RESUME." Since performance graphics, especially the jamming variety, requires constant real-time debugging with two hundred people looking over your shoulder, CONTROL-S comes in very handy. The combination of background DOLOOPED macros and CONTROL-S give the user the impression he is always in control.

ERROR MESSAGES AND DEBUGGING IN GRASS

Grass has about one hundred fifty error messages, only one of which is truly cryptic ("Undiagnosable syntax error"). When an error occurs, the whole command is printed out on the VT05 with a little arrow under the part that caused the error, followed by the error message. The user is then put into the same mode that CONTROL-S initiates, at which time he can correct the mistake and RESUME or type CONTROL-C to abort. Along with the feedback on the screen, these error messages and interactive fixups account for about ninety percent of the user debugging activity.

The harder-to-find logic errors are usually tracked down by the LIST command (by GM), the TRACE command (by TO) or sometimes the DOLOOP code. LIST simply prints each line of a macro as it executes. TRACE, adopted from SNOBOL, takes variable names as arguments and then prints out the value of the variables every time the variables are changed along with the location of the change. As a last resort, a macro to sense an elusive logic error may be parallel processed with the defective macro to discover the problem. For instance, assume variable A is never supposed to get to zero but it is anyway. "DOLOOP <IF A EQ 0,LIST>" will turn on the LIST feature as soon as A goes to zero. Obviously, more exotic bug traps can be constructed.

Error conditions may also be trapped and further processed by the macro writer. The ONERROR command sets up an asynchronous error recovery procedure for use when an error happens:

```
ONERROR VARIABLE,ANY GRASS
  STATEMENT
```

When the error occurs, the error number is put in the variable indicated for use by the error recovery routine and the Grass statement following the comma is executed in place of the line in error. Since errors can be as benign as not finding a file on the disk, system macros are written with ONERROR frequently to help the novice user. As might be imagined, considerable detail work was done to assure PDP-11 stack integrity when commands in error are replaced by ONERROR code. Note that the Grass statement in the ONERROR command can be a macro call like "DO FIXUP."

The expert user can take advantage of the asynchronous nature of the ONERROR command to speed up loops. For example, in the following program which zeros the z-value of each vector, the end-of-picture condition (K= −1) must be tested:

```
ZEROZ:<PROMPT "NAME OF PIX FOR Z-AXIS
  ZEROING"
INPUT $L
N=0
N=N+1
GETPOINT $L,N,X,Y,Z,K      (set the nth point in
                            variables x,y,z and k)
IF K NE −1,ZAPPOINT $L,N,X,Y,0,K; SKIP −2
                   (if K= −1, it is the end of picture)
PROMPT "DONE ZEROING">
```

Now, if "ONERROR A,SKIP 2" is placed somewhere before the last four lines of this macro, the test for K= −1 may be eliminated because the GETPOINT index (N here) will go out of bounds and generate an error. "SKIP 2" will be executed in place of the line in error and the control will pass to the last PROMPT. Such tricks can even work with compiled macros, a housekeeping feat of some proportion.

In summary, Grass provides many ways of controlling, debugging and interacting with pictures and macros. Since so much of the control can be parallel, the user occasionally feels like he is conducting rather than watching a plotter. Music has always been a performing art and artists now have the tools to perform visual scores. The task now is achieving the control subtlety in performance graphics that we know and love in music.

CURRENT LIMITATIONS OF GRASS

Grass as a system has for some time been pushing against walls created by equipment speed, memory limitations and the nature of the refresh display. Currently, the PDP-11/45 we use has only 28K of usable memory, of which 3K is used by the disk operating system. The user is left with 10K of space to use. Given the software overhead, the maximum number of parallel full-screen vectors that can be displayed flicker-free is about a thousand, although they can all be rotating, moving and scaling. We have to operate in a flicker-free environment because our television-based system is not sophisticated enough to operate in anything but real-time.

Adding speed to the PDP-11/45 is fairly simple with an add-on cache memory, but not cheap. Adding to the memory requires memory management, and a considerable amount of reprogramming. And CRT's that are faster by an order of magnitude are still a gleam in the designer's eye.

A major limitation to the habitability of Grass is that not all users have linguistic skills—or type well. Non-linguistic approaches to subset problems like constructing complex 3-D pictures are possible using light-pen or tablet menus. We now have several small grants to investigate performance-time control structures for computer graphics.

REFERENCES

1. DeFanti, T. A., Dissertation, The Ohio State University, 1973.
2. DeFanti, T. A., D. J. Sandin and T. H. Nelson, "Computer Graphics as a Way of Life," *Computers & Graphics*, Vol. 1, No. 1, May 1975.
3. DeFanti, T. A., "Toward Loopless Interactive Graphics Programming," *Proceedings of the Conference on Computer Graphics, Pattern Recognition and Data Structures*, May 14-16, 1975 (IEEE Catalog #75CH0981-1C).
4. DeFanti, T. A., D. J. Sandin, et al., Interactive Electronics Visualization Event, Videotape, 90 mins., OIRD University of Illinois at Chicago Circle, 1975.
5. Baecker, R. M., Dissertation, M.I.T., 1969.
6. Nelson, T. H., *Computer Lib/Dream Machines*, 1974.

7. Newman, W. M. and R. F. Sproull, *Principles of Interactive Computer Graphics,* McGraw-Hill, 1973.
8. Vector General Inc., *Graphics Display System Reference Manual,* 1975.
9. Csuri, C., et al., *Real-Time Film Animation,* The Ohio State University Research Center, February 1973.
10. DeFanti, T. A., Grass Internal Logic, Videotape, 120 min., OIRD The University of Illinois at Chicago Circle, 1974.
11. Donato, N. and T. A. DeFanti, 2-D and 3-D Transformations, Videotape, 25 min., OIRD The University of Illinois at Chicago Circle, 1975.
12. Collins, D. W., "A Computer Graphics System for Modular Building Elevation Design," *Proceedings of the Second Annual Conference on Computer Graphics and Interactive Techniques*—SIGGRAPH '75, July 1975.
13. Personal communication.

Braille grade II translator program

by MONIQUE TRUQUET*
Université Paul Sabatier
Toulouse Cedex, France

ABSTRACT

This paper describes the Computer Translation of French inkprint into Grade II. It will be a reminder of what I have presented during two workshops and a conference, increased by the last modifications.

THE FRENCH GRADE II

The Translator program was written in Basic FORTRAN IV two years ago and it has undergone some modifications. We are going to present the method used. To recognize EXPRESSIONS like c'est-à-dire (that is to say), to recognize WORDS, CONTRACTIONS, NUMBERS—we use a syntactic analyzer with a list structure and a hashing table. The organigram presented (Figure 1) shows the different ways of the program. The grammar used follows. We have chosen for axiom TEXT and the grammar is written on the Backus Normal Form.

TEXT::=(TERM Σ TERM) followed by (PUNCTUATION MARK Σ PUNCTUATION MARK)
TERM::=SPECIAL SIGN/EXPRESSION/WORD/NUMBER
WORD::=LETTER Σ LETTER
NUMBER::=DIGIT Σ DIGIT
SPECIAL SIGNS are in a TABLE the form of which is a Tree
EXPRESSIONS are in a TABLE the form of which is a Tree
LETTERS are in a TABLE
DIGITS are in a TABLE
PUNCTUATION MARKS are in a TABLE

Expressions

It is the grouping "Pour ainsi dire" which is translated but not Pour (for) then AINSI (so) then DIRE (to say). To find an expression let us see in Figure 2 that the checker has to know if the first letter is or A or C or D or E or L or N or P or W or S or T or V. In supposing that it is A, we have to survey the "Tree" in Figure 3.

If no expression is found then the checker search is to isolate a word.

(a) *with A we can found*:

Au contraire	→(on the contrary)
Au-dessous	→(below
Au-dessus	→(above)
Aujourd-hui	→(to day)
Autant que	→(as far as)
Autant qu'	→(as far as)
Autre chose	→(something else)
à part	→(except for)
à cause	→(on account of)
à mesure	→(in proportion as)
à peine	→(hardly)
à peu près	→(nearly)
à présent	→(now)
à travers	→(through)

(b) *with P*

Parce que	→(because)
Parce qu'	→(because)
Par conséquent	→(therefore)
Par exemple	→(for example)
Par suite	→(consequently)
Par-dessous	→(under)
Par-dessus	→(on top of)
Peu à peu	→(little by little)
Peut-être	→(perhaps)
Plus tard	→(later)
Plus tôt	→(earliest)
Pour ainsi dire	→(so to speak)

The tree which permits us to find the expressions which begin by the letter A is represented in Figure 3.

Words

Supposing that a word is recognized then a hashing method is applied and the word is searched in the

* Centre d'Informatique de Toulouse (France).

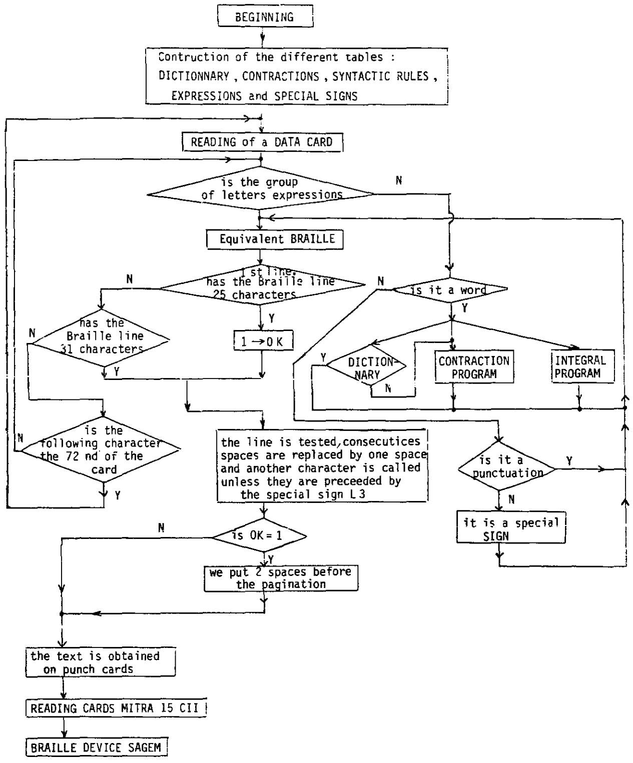

BEGINNING
Contruction of the different tables : DICTIONNARY , CONTRACTIONS , SYNTACTIC RULES , EXPRESSIONS and SPECIAL SIGNS
READING of a DATA CARD
is the group of letters expressions
N
Equivalent BRAILLE
1 st line. has the Braille line 25 characters
N
Y
has the Braille line 31 characters
N
Y
1 → 0 K
is the following character the 72 nd of the card
N
Y
the line is tested, consecutices spaces are replaced by one space and another character is called unless they are preceeded by the special sign L 3
is OK = 1
N
Y
we put 2 spaces before the pagination
the text is obtained on punch cards
READING CARDS MITRA 15 CII
BRAILLE DEVICE SAGEM
is it a word
N
Y
DICTION-NARY
Y
N
CONTRACTION PROGRAM
INTEGRAL PROGRAM
is it a punctuation
Y
N
it is a special SIGN

Figure 1

Braille dictionary. If it is, the program takes the equivalent Braille if not the program search of the word is composed with contractions.

Difficulties encountered

(1) *To obtain a Braille Page correctly:* the program has to obey precise rules and it has to replace consecutive spaces with only a space.

(2) *to use the contraction method:* if some groups of letters are forgotten the program cannot continue; it was the case for the groups:

AI3→(aî)
AI2→(aí)

but the correction is easy.

(3) *to treat homonyms:* (we must specify to the program those which obey the rules that they will be preceded by the special sign "::").

- like CONVIENT:

I am spelling this word because it has not the same pronounciation; it depends on the meaning. This could be a form of the verb "CONVIER" (to invite). It is pronounced CONV(I) or the verb "CONVENIR" (to agree with) is pronounced

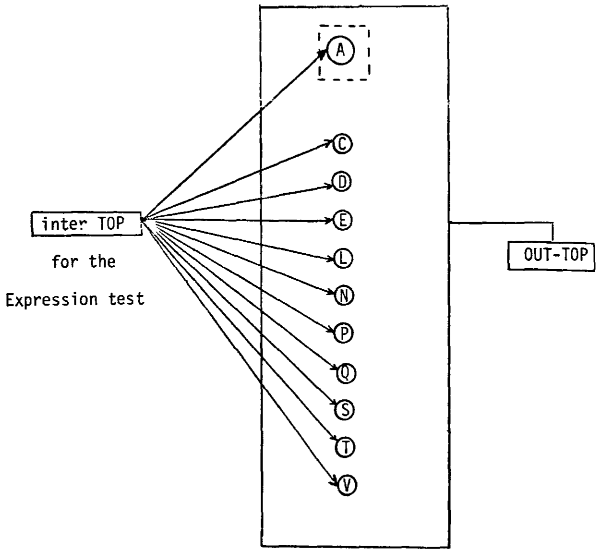


Figure 2

CON VI (IN). If it means to invite, the Braille translation is (CON)-V-I-(ENT). If it means to agree with, the Braille translation is (CON)-V-(IEN)T. "To agree" obeys the rules and with the sign mentioned above the program directs this word to the contraction table.

- like FILS:

If it is the "SON" it is pronounced (FIS), the Braille translation is F-S. If it is the "threads"

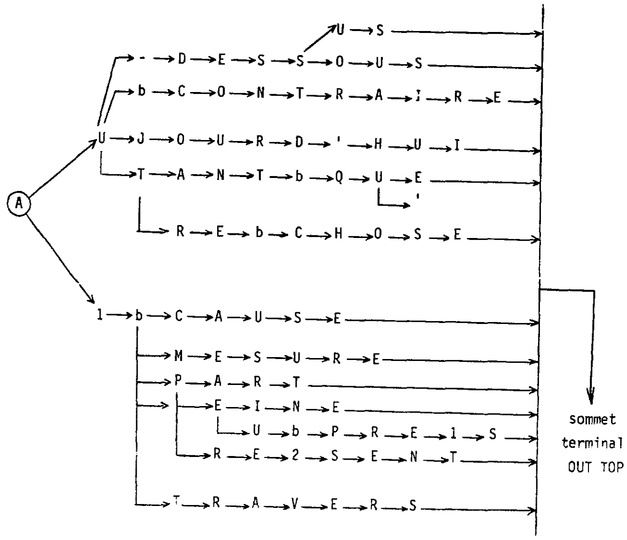

A
U → S
- → D → E → S → S → O → U → S
b → C → O → N → T → R → A → I → R → E
U → J → O → U → R → D → ' → H → U → I
T → A → N → T → b → Q → U → E
'
R → E → b → C → H → O → S → E
1 → b → C → A → U → S → E
M → E → S → U → R → E
P → A → R → T
E → I → N → E
U → b → P → R → E → 1 → S
R → E → 2 → S → E → N → T
T → R → A → V → E → R → S
sommet
terminal
OUT TOP

Figure 3

it is pronounced (FIL) and the Braille translation is F-I-L-S.

(4) *to write foreign words:* foreign words and proper names must be written in grade I preceded or with a capital, or with the special sign which shows to the blind that the word which follows is in grade I.

(5) *to know if a word must be translated or in grade I or in grade II.* It is the case for the words SI (if) or PUIS (then) or CELUI (this one or that one). When they are not followed by a punctuation mark they can be translated in grade II if not they must be translated in grade I and preceded of course by the special sign "integral".

MODIFICATIONS

We have seen that a part of the words put in the dictionary can be translated by the contraction method. This is interesting because we have only one word to add to the contraction table and with the prefixes, suffixes of this table we can produce all the words of the same family.

With the word "EGAL" (EQUAL) written E2GAL we can create

E2GALE
E2GALES
INE2GAL→(IN)-(E2GAL)
INE2GALE→(UNEQUAL)
INE2GALES

but we cannot obtain

E2GALITE2	→E-G-T	(equality)
E2GALITAIRE	→E-G-T-R	(equalitarian)
E2GAUX	→E-G-X	(equal)
E2GALEMENT	→E-G-M	(also)

and these words stay on the dictionary because they don't obey the contraction rules.

With the word "POINT" we can create about 40 words so it is very interesting.

Here are some:	POINTE	head, touch
	POINTU	sharp
	POINTAGE	checking
	POINTEAU	needle
	POINTS	points
	POINTER	to check

Remark: If we want to obtain the words E2GALITE2, E2GAUX by the contraction method we must change the sense of the test of the word. At the present time we begin by the right as some suffixes are longer than prefixes, but for the new method it will be necessary to begin by the left to isolate first the prefix, then the root; but will the Braille text obtained be more correct than now because of some roots which begin by the same letters than some prefixes?

CONCLUSION

This year we have a Braille device which satisfies us so we have the care to translate texts for a special school for blind (ASEI)* and for students of the Letter University. French grade I is very important for young children, grade II for the others, and the special schools need translations. Students of our University need above all Braille mathematics and we are going to write the program. Later we'll try to create Informatic trainings for the blind at our Informatic Center of Toulouse. At the present time we work more easily thanks to a French organization IRIA** which sent us some money to pay an Engineer and the making of a French Braille device built by the society SAGEM‡ so that it will be better in the future.

REFERENCES

1. Association Valentin Haüy, Abrégé Orthographique étendu, (Extended Contracted Orthography), Edition 1964, 9 rue Duroc, PARIS 75007.
2. Espitallier, J., Abrégé Orthographique Français étendu, (Contracted French Orthography), Lycée Parc St Agne Ramonville-Toulouse 8 October 1966.
3. Kahan, B. and H. Dumas-Primbault, "Principe d'une méthode syntaxique d'écriture de compilateurs," (Principles for a syntactic method for scripture of compilators), *Association Française pour la Cybernétique Economique et Technique,* (French Association for the Economical and Technical Cybernetics), August 1969, No. B2.
4. Bloch, M. and C. Brisseau, "Analyse syntaxique par cheminement sur un graphe," (Syntactic Analysis by graphical methods), *Association Française pour la Cybernétique Economique et Technique,* (French Association for the Economical and Technical Cybernetics), March 1970.
5. Truquet, M., "Automatic translation into Braille, (grade 2)," Paper presented to a Workshop in Münster (RFA) March 1973.
6. Truquet, M., "The automatic transcription of French ink print into Braille," *Research Bulletin,* No. 28.
7. Truquet, M., *Some remarks on the French Braille Translation,* Lecture given in the Research Center of Münster (RFA), September 1974.
8. Truquet, M., "Automatic Translation into Braille in Basic Fortran IV," Paper presented to the Second Workshop in Copenhagen (Denmark), September 1974.

* ASEI: Association pour la Sauvegarde de l'Enfance Invalide (Association for the Safeguard of the Invalid Childhood)

** IRIA: Institut de Recherche d'Informatique et d'Informatique (Institut of Informatic and Automatic Research)

‡ SAGEM: Société d'Applications Générales d'Electricité et de Mécanique (Society of General Applying for Electricity and Mechanical).

Interfacing computers for the physically handicapped—A review of international approaches

by GERALD A. RAITZER, GREGG C. VANDERHEIDEN and CRAIG S. HOLT
University of Wisconsin
Madison, Wisconsin

ABSTRACT

The computer has shown great promise for expanding educational and occupational opportunities for severely handicapped individuals. The greatest obstacle facing these individuals today is that they are unable to access computers through traditional keyboards or other types of terminals. This paper describes the three basic techniques which have been successfully used to allow the severely handicapped to communicate with other people and with their environment. A review of over 25 various approaches which have been developed around the world is presented, including a comprehensive bibliography of available information.

INTRODUCTION

For many individuals severe physical handicaps have completely cut off most avenues of personal development and employment. Their physical involvement bars them from any constructive or creative activities requiring physical or manipulative abilities. Moreover, their inability to speak, write or efficiently operate even simple communication devices severely impairs their ability to develop and exercise their mental capacities. This latter problem is basically an output problem in which a normally functioning intellect is trapped within a body having no effective means of communicating or interacting with the environment. Fortunately with today's technology, especially micro electronics and the computer, new avenues are being opened for these individuals which promise them not only a chance for a more effective education and a more meaningful mode of self expression, but also a means of self support through employment.

The major problem in trying to realize the full potential of these individuals is in finding efficient means of communication for them. Information output should consist of both written communication and discrete commands with which they can control certain elements or devices in their environments.

Two years ago we presented one approach for providing the severely physically handicapped with a means to control computers and other data and information processing equipment. The approach described was developed for individuals who had some pointing capability but who were physically unable to use regular keyboards in an effective manner. In this paper we will be presenting a summary of all of the different basic approaches which have been developed around the world to interface individuals with varying physical abilities. With these different approaches most any individual, no matter how severe his physical handicap, could be provided with an effective means for controlling communication and other interactive devices.

THREE BASIC APPROACHES

Although a great many different aids have been developed to provide the physically handicapped with a means of controlling other devices, their control schemes are all essentially variations on three basic approaches. These approaches are scanning, encoding and direct selection. Each of these approaches has different advantages and disadvantages and will work better with one kind of disability than with others. Some of the techniques such as scanning are much more powerful in that they can be used by even the most severely handicapped. Other techniques may be less powerful but more efficient, thus allowing individuals who have more control over their movements to be able to communicate in a faster manner by taking advantage of their better physical abilities.

THE SCANNING APPROACH

The definition of scanning aids which has been developed by the Trace Center (University of Wisconsin-Madison) is:

Any technique or aid in which the selections are offered to the user by a display and where the user selects the characters by responding to the display. Depending upon the device, the user may respond by simply signaling when he sees the correct choice presented or by actually directing the indicator (e.g. light or arrow) toward the desired choice.

One example of a scanning aid would be a simple rotating pointer (see Figure 1(a)). With such a system the individual could start and stop the arrow using some switch that was specially adapted to take advantage of some movement over which he had good control. He could then use the arrow to point to the various characters desired. An encoding wheel on the back could digitize the selection process for entry to the computer. This type of scanning is referred to as linear scanning. A more efficient technique which has been developed is a two-speed linear scan whereby the individual can operate one button which will cause the aid to scan rapidly until it approaches the correct choice. The individual then releases the button and the aid slows down to enable an accurate selection by the individual.

Another technique which can be used to increase the scanning speed is an XY or row column scan (see Figure 1(b)). Here the aid lights one entire row at a time until signaled by the user. The aid then lights up the entries in that row one at a time until the user signals it again. The character is then output and the aid resumes scanning.

For individuals who can control a joystick or other set of four switches, a directed scanning technique can provide an even more efficient scanning control system. With this system the individual can direct the lighted square up, down, left or right to move it to the desired character. He can then release the control switch allowing the light to remain on the square for a predetermined time interval after which it is output. This technique has also been implemented using switches where one is pushed alternately to control the up and down direction and the other is pushed alternately to cause the light to move left and right.

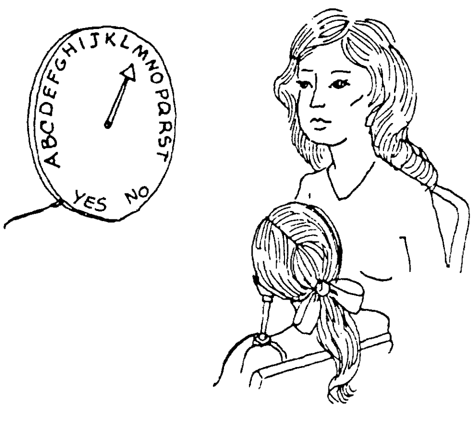


Figure 1(a)—Simple linear scanning technique

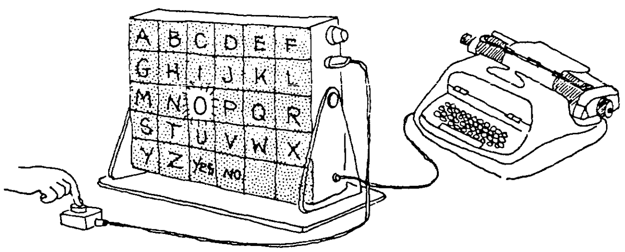


Figure 1(b)—Row/column scanning technique

The latest technique to be developed in the scanning category is the "anticipatory scan" or computer assisted scan. With this technique the aid looks at the last one or two letters which have been printed, refers to its programmed memory for the next most probable letters, and then displays those letters first. For example: if the last two letters printed by the handicapped individual were "...TH", the aid would present the letters E, A, SPACE, I... and then scan over the remaining letters afterwards. In this manner the letters which are most likely to occur next are presented to the individual first, increasing his speed for selection. Using this system the odds are about 80% that the next letter that the individual wants will be one of the first five characters presented by the anticipatory scanning aid.[7]

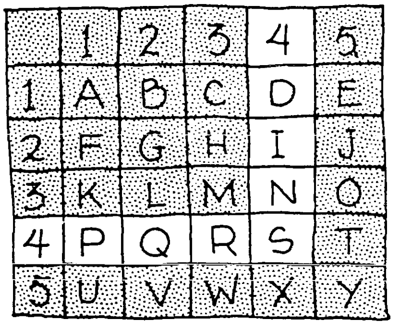
THE ENCODING APPROACH

The definition adopted for encoding aids is:

> Any technique or aid in which the desired characters are indicated by a pattern or a code of input signals, where the character codes must be memorized or referred to on a chart. Any number of switches may be used (e.g., 1, 2, 4, 7, 8, etc.). The code may involve activating the switches sequentially, simultaneously or in a specific time sequence.

One example of an encoding technique would be an aid where two numbers given in sequence are used to encode the desired characters. For instance, in the simplified example shown in Figure 2(a) a "2" and a "3" could stand for an "m" and a "4" and a "4" would stand for an "s". By using a larger number of encoding numerals (e.g., 10 or 12), larger numbers of characters and control commands can be encoded while keeping the number of keys relatively few.

For individuals who have very poor gross motor movement but who do possess some very fine motor control, techniques such as Morse Code may be more efficient. (Figure 2(b)). Here an aid can take advantage of a very small muscle movement or even a bio-potential to control a device which could decode



	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I	J
3	K	L	M	N	O
4	P	Q	R	S	T
5	U	V	W	X	Y

Figure 2(a)—Simplified example of a two-number encoding scheme

his movements and in turn control some form of output or data processing equipment.

THE DIRECT SELECTION APPROACH

A direct selection technique is:

> Any technique or aid in which the desired output is directly indicated by the user. In direct selection aids there is a key or sensor for each possible output selection.

This is the most common form of control used in the computer industry. All keyboards would fall into this category. For the severely handicapped, a number of guarded and expanded keyboards have been developed (Figure 3(a)). For individuals who have some pointing abilities, but whose motions are too erratic to control even enlarged keyboards, other special techniques have been developed which can interpret even very erratic pointing motions. These techniques would also fall under the category of direct selection aids.

PROFILE OF EXISTING COMMUNICATION AIDS

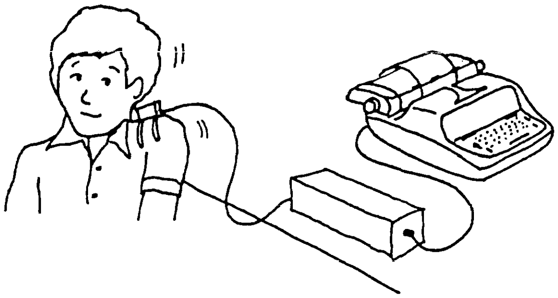


Figure 2(b)—Small muscular movement used to control output device via encoding technique

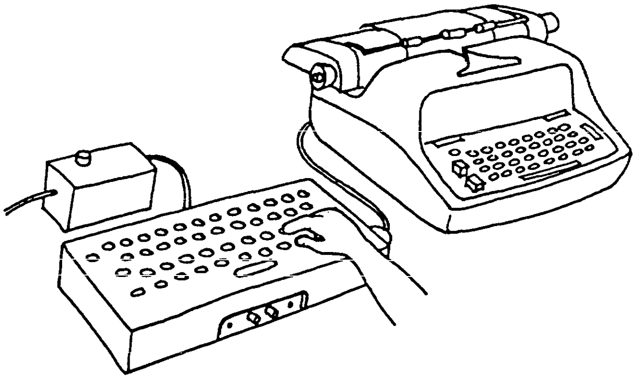


Figure 3(a)—A direct selection technique in the form of an expanded recessed keyboard

The following is a descriptive survey of aids currently available or under development in each of the above categories. This descriptive survey does not include all of the aids which exist in each of these categories. A more comprehensive listing and description of these and other aids is available from the Trace Center, University of Wisconsin-Madison. (See Bibliography).

SCANNING COMMUNICATION AIDS

The scanning type of communication aid is currently the most common form of communication aid available. The major advantages of these aids are: (1) they require little physical effort to operate and (2) they can be used by most anyone no matter how severe their physical disability.

The major disadvantage of these aids is their slow speed. The user must wait until the indicator stops at many unwanted characters before the aid will reach the desired character. The rate of scanning must stay relatively slow because the indicator must rest on each selection long enough for the operator to select it without error.

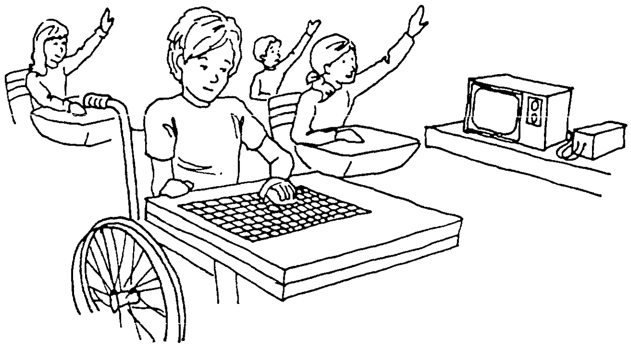


Figure 3(b)—Auto-monitoring technique; a direct selection technique which can be used by individuals having only very erratic pointing skills

Many different variations of this technique have been developed. A device which scans by eliminating the characters in a row column fashion was developed for use with severe myasthenia gravis patients by the Department of Surgery of the Royal Post Graduate Medical School, (London, England). Centre Industries, (New South Wales, Australia) also has an illuminated letter scanning device which uses a linear scan to control an electronic typewriter. A two-speed linear scanning aid is available from PMV (Stockholm, Sweden) which controls a standard IBM typewriter using a solenoid matrix positioned over the keyboard.

Several groups have developed aids which utilize the row column scanning pattern. With these aids a larger number of selections can be displayed without requiring great amounts of time to select them. The increased efficiency, however, requires two signals to be given for each letter selected. The Tufts Interactive Communicator (TIC) developed at Tufts University, (Boston, Massachusetts) uses the row column scanning technique and has a Burroughs display and strip printer as two of its output forms. Other versions of this aid have been adapted to work with a variety of outputs. The "Linguaduc Scanner" produced by Carba (Switzerland) uses one or two switches to control a Facit printer. POSSUM Controls, Limited (Aylesbury, England) produces the "POSSUM" device which uses an adapted typewriter as its output form, as does the Zambette Electronics (Essex, England) "System 8" unit. The POSSUM typewriter features a simple sip-puff switch and chin switch while the System 8 unit features a "magic" capacitance switch and projection system for displaying typewriter output on the wall or ceiling. The "Comhandi" produced by Physio-medical Systems (Montreal) is built into the base of a Teletype which it uses as its output form. The Cybernetics Research Institute (Washington, D.C.) also has a scanning communication aid called the Whispertype which controls a typewriter and can be used with a variety of input sensors. This aid has also been recently interfaced with a voice synthesizer. The Alphabet Message Scanner developed by Prentke-Romich (Shreve, Ohio) is a very simple row column scanning device which runs on batteries and has a data output jack on its back for control of other devices. A completely portable scanning aid has been developed by Vendacom (Brooklyn, New York) called the Porta Printer. This aid is built into an attaché case and weighs just 17 pounds. It uses a miniature strip printer as its output and can control two electrical outputs for the convenience of the user. The STRIP PRINTER is an aid which is very similar though lighter than the Porta Printer and is also marketed by Prentke-Romich (Shreve, Ohio).

Another fully portable scanning aid, which is built into a special lap board, has been developed by the Trace Center (Madison, Wisconsin). This aid has several output forms including typewriter, television display, teletype and voice synthesizer in addition to its built in strip printer and correctable alpha-numeric display.

As mentioned earlier, the anticipatory scanning technique was first introduced by the New England Biomedical Engineering Center at Tufts University (Boston, Massachusetts) and is now being incorporated into the newest version of the Tufts Interactive Communicator (TIC). A computer assisted anticipatory scanning system has also been implemented at Northwestern University, Rehabilitation Engineering Center (Chicago, Illinois) using a DEC mini computer.

One of the principal advantages of the scanning approach that has been mentioned is the fact that it can be used by almost any individual no matter how severe his handicap. This is due to the fact that the operator may use a single input switch to control the aid. In addition, any one of a large number of input switches or sensors could be used to provide the signal. Some examples of switches that may be used are breath switches, knee switches, impact switches, pull switches, proximity switches, whistle switches, the NASA Eye Movement Switch, eyebrow switches, muscle potential switches and brainwave sensors, to name a few. Thus, at least one switch could be found which could be operated by a given individual.

ENCODING AIDS

Because speed is so important when communicating (especially character by character) several research groups have developed devices that use encoding techniques instead of the scanning approach. The increased efficiency of signaling provided by the encoding approach increases the speed of communication. Here again, however, the increased speed requires a greater degree of control from the operator. More complex motions and sometimes more responses per letter are required. In addition, the encoding scheme must be learned before the device can be used. The greater the number of switches used in the encoding scheme, the simpler the code will be. However, an increased number of switches requires greater dexterity on the part of the operator. Thus, a compromise between the two factors, simplicity of code and number of switches, must be worked out. Different research groups have chosen different balances between the two factors in the design of their aids. The aid best suited to an individual is determined by his specific communication needs, and his physical abilities.

Centre Industries (Australia) chose a simple set of movements (two levels and a pause) as a Morse Code, to operate their COS unit which in turn controls a typewriter. Medicel (South Burlington, Vermont) also uses a Morse Code communication aid. This device, called the MC 6400, uses a television monitor as its output form. POSSUM Controls (Aylesbury, England) uses a three level code in one of their POSSUM units (sips, pauses, and puffs) which also controls a typewriter. The aid developed by Hengrove, Ltd. (Berkshire, England) uses a four level code (two degrees of sips and two degrees of puffs) which requires finer breath control but is faster. With the Electraid, 1, 2, 4 or 8 switches can be used to control a typewriter. The Cybernetics Research Institute (Washington, D.C.) uses a still larger number of switches in their CYBERTYPE device. The CYBERTYPE uses an encoding scheme which utilizes 14 switches arranged in two banks of 7. One switch in each bank must be selected to print a character. The aid may also be used with only 7 keys by just pressing two keys in succession. A portable independent encoding aid has been developed at the Trace Center (Madison, Wisconsin). This aid is built into a special wheelchair laptray and features correctability and the ability to print out words and phrases in addition to individual characters.

The basic advantage the encoding devices tend to have over scanning systems is the greater speed—a most important factor. Encoding devices do however require that the operator have considerably greater dexterity. They also require that the operator learn a code of some sort before operating the device. Those who have used encoding aids however report that the codes are usually easily learned and retained by the individuals using them.

DIRECT SELECTION AIDS

Communication aids in this category have the greatest number of switches; one for each symbol or character on the aid. To obtain a character the user simply operates the switch corresponding to that letter and that letter only. One obvious advantage with this type of aid is that its operation is very simple and straightforward. These aids also tend to be more rapid than the scanning systems since there is no need to wait for a scanning indicator to arrive at the proper letter. Speed of communication is limited only by the user's ability to activate one of the aid's switches. The major limitation of aids in this category is that they require increased dexterity on the part of the operator, who must have fairly good control of the gross motor movements of at least one of his extremities.

Some of the aids in this category are designed for persons with no arm or hand control but fairly good control of other portions of the body. MEFA GmbH (Bonn, Germany) makes an expanded four tier keyboard for operation with one's feet. Reva-Aids (Copenhagen, Denmark) makes an expanded proximity keyboard for use with cerebral palsied individuals. This keyboard can be operated by either foot or by headsticks. IBM makes special handrests and armrests which allow some weak and spastic individuals to operate standard IBM electric typewriters. When used with a correctable IBM Selectric typewriter these adaptations can provide the handicapped individual with a correctable communication aid. Furthermore a special "ORATOR" type-ball is available which can provide large and highly visible characters for the vision-impaired. Several special keyboards have also been developed by PMV (Stockholm, Sweden) to enable use of the typewriter by the severely handicapped.

A typewriter that is controlled by a light pen attached to the user's head has been developed by M. Soede and H. G. Stassen (Netherlands) and is called a LOT (light operated typewriter). Several telecommunication aids for the deaf can also be modified for the moderately involved individual, including the MCM Communicator marketed by SICO (Oakland, California) and the TV Phone marketed by TV Phone (Silver Spring, Maryland). A hand held communicator which can be worn on the wrist has been developed by Cannon, Incorporated (Japan) in cooperation with researchers in Holland. The aid is called the Cannon Communicator and uses a strip printer as its output. A portable direct selection communication aid which is built into a laptray has been developed by the Trace Center (Madison, Wisconsin). This unit, called the Auto Monitoring Communication Board (Auto-Com) has a special sensing system which allows the aid to be used by individuals having only very minimal pointing skills. The aid has a smooth, flat, hard surface and operates in much the same manner as a traditional communication board except that the user's output is automatically printed out for him on a self-contained strip printer or other output device such as a typewriter, CRT display or teletype. The Auto-Com is capable of printing entire words, phrases or strings of commands in addition to being able to print out individual characters.

CONCLUSION

It can be seen that a large number of devices have been developed for the physically handicapped, although only a limited number of these aids have been developed to interface with computers. All of the techniques described could be easily adapted to provide access to computers for the physically handicapped. It can also be seen that because of the diversity of approaches and specific techniques which have been developed by the various researchers, it should be possible to select an aid which could provide any particular individual with a control scheme which would take optimum advantage of his physical skills. Thus, the inability to control traditional data entry devices should no longer be a barrier to the use of computers by the severely handicapped. Moreover, two other features of the computer, remote terminals and time sharing, can help to overcome two other barriers, (transportation problems and the decreased work speed of more severely handicapped individuals) that may interfere with the physically handicapped individual's ability to participate in other types of endeavors.

The potential for the use of the computer in the instruction of the handicapped (particularly the more severely handicapped whose slower response time can make optimum use of the "patience" of the computer) should also not be overlooked.

FOR FURTHER INFORMATION

A master chart of communication aids listing the above communication aids and others in a chart form profiling their features, as well as an annotated bibliography of communication aids providing pictures, descriptive information and where to secure additional information on each of the aids, are both available from the Trace Research and Development Center for the Severely Communicatively Handicapped, University of Wisconsin-Madison, 922 ERB—1500 Johnson Drive, Madison, Wisconsin 53706.

ACKNOWLEDGMENTS

A special note of gratitude to Mary Jo Luster whose efforts are responsible for the location and classification of many of the aids contained in this summary. We would also like to thank the many individual researchers and groups who have assisted the Trace Center in compiling these summaries for use by other concerned professionals.

REFERENCE BIBLIOGRAPHY

1. Boydell, R. G., "Possum—I Can," *Spastic News*, May 1964, p. 6.
2. Brulisaur, Peter, "Mailing Rehabilitation Systems," *Proceedings of the Seminar on Electronic Controls for the Severely Disabled*, Vancouver, British Columbia, Canada, pp. 37-39, 1974.
3. Bruun, Georg, Bjorn Jarkler, Peter Speldt and Arne Nybroe Sorensen, "Communication Display/Printout Aid with a Memory," In *Aids for the Severely Handicapped*, pp. 116-118. Edited by Keith Copeland. London, Sector Publishing Limited, 1974.
4. Charbonneau, J. R., C. Cote and O. Z. Roy, "NCR's 'Comhandi' Communication System, Technical Description and Application at the Ottawa Crippled Children's Treatment Centre," *Proceedings of the Seminar on Electronic Controls for the Severely Disabled*, Vancouver, British Columbia, Canada, pp. 83-88, 1974.
5. Copeland, Keith, *Aids for the Severely Handicapped*, London, Spector Publishing Co., Ltd. 1974.
6. Crochetiere, J. W., R. A. Foulds and R. G. Sterne. "Computer-Aided Motor Communication," *Proceedings of the 1974 Conference on Engineering Devices in Rehabilitation*, Boston, Massachusetts, pp. 1-5, 1974.
7. Foulds, Richard A. and Earl Gaddis, "The Practical Application of an Electronic Communication Device in the Special Needs Classroom," *Proceedings of the Seminar on Devices and Systems for the Disabled*, Philadelphia, Pennsylvania, pp. 77-80, 1975.
8. Foulds, Richard A., Gregory Baletsa and William J. Crochetiere, "The Effectiveness of Language Redundancy in Non-Verbal Communication," *Proceedings of the Seminar on Devices and Systems for the Disabled*, Philadelphia, Pennsylvania, pp. 82-86, 1975.
9. Foulds, R. A., "The Tufts Interactive Communicator," *Proceedings of the 1972 Carnahan Conference on Electronic Prosthetics*, Lexington, Kentucky, pp. 16-24, 1972.
10. Hyde, C. David, The Possum Selector Unit Type I, Aylesbury, P.O.S.M. Research Project.
11. Iles, G. H., "Interfaces for the C.P.," *Proceedings of the Seminar on Electronic Controls for the Severely Disabled*, Vancouver, British Columbia, Canada, pp. 71-82, 1974.
12. Jefcoate, Roger, "Possum—Its Significance to Multiple Sclerosis Patients," *M.S. News*, Christmas, 1970.
13. Jenkin, Rosemary, "Possum: A New Communication Aid," *Special Education*, March, 1967, pp. 9-11.
14. Jones, Morris Val, "Electrical Communication Devices," American Journal of Occupational Therapy, 15, May-June 1961, pp. 110-111.
15. Luster, Mary Jo, *Selected Bibliography of Articles, Brochures and Books Related to Communication Techniques and Aids for the Severely Handicapped*, Trace Center, 922 ERB, University of Wisconsin-Madison, Madison, Wisconsin 53706, 1975.
16. Luster, Mary Jo, *Annotated Bibliography of Researchers and Institutions*, Trace Center, 922 ERB, University of Wisconsin-Madison, Madison, Wisconsin 53706, 1975.
17. Luster, Mary Jo and Gregg C. Vanderheiden, *Annotated Bibliography of Communication Aids*, Trace Center, 922 ERB, University of Wisconsin-Madison, Madison, Wisconsin 53706, 1975.
18. Mailing, R. G. and D. C. Clarkson, "Electronic Controls for the Tetraplegic (Possum)," *Paraplegic*, March 1963, pp. 162-174.
19. Miller, John, "Electronics for Communication," *American Journal of Occupational Therapy*, 18, January-February 1964, pp. 20-23.
20. Possum Controls, "General Descriptions and Specifications: P.O.S.M. Research Project," Aylesbury, 1967.
21. Rahimi, Morteza Amir and John Bryson Eylenberg, "A Computer Terminal with Synthetic Speech Output," A paper presented at the National Conference on the Use of On-Line Computers in Psychology, St. Louis, Missouri, 31 October 1973.
22. Rahimi, Morteza Amir and John Bryson Eylenberg, "A Computing Environment for the Blind," *AFIPS Conference Proceedings of the 1974 National Computer Conference*, Vol. 43.
23. Tolstrup, I. M., VIDIALOG: TV-Based Communication System for the Motor Handicapped, *Proceedings from III Nordic Meeting on Medical and Biological Engineering*, Tampere, Finland, 1975.
24. Vasa, J. J. and M. Mansell, "Queen's Communication Aids," *Proceedings of the Seminar on Electronic Controls for the Severely Disabled*, Vancouver, British Columbia, Canada, pp. 37-39, 1974.
25. Kafafian, H., *Study of Man-Machine Communication Systems for the Handicapped*, 3 vols. Washington, Cybernetics Research Institute, Inc., 1970-1973.
26. Newell, A. F. and C. D. Nabaui, "VOTEM: The Voice Operated Typewriter Employing Morse Code," *Journal of Scientific Instruments*, Series 2, 2, 1969.
27. Englehart, T. W., "A Computerized Typing System for the Handicapped," M.S. Thesis, University of Alberta, 1971.
28. Kildaw, R., "A Multi-Terminal Interface Allowing Typewriter Operation by Paraplegics (MINTOP)," M.S. Thesis, University of Alberta, 1968.
29. Soede, M. and H. G. Stassen, "A Light Spot Operated Typewriter for Severely Disabled Patients," *Medical and Biological Engineering*, 1973, pp. 641-644.

30. Stassen, H. H., M. J. Soede and W. J. Luitse, "The Light Spot Operated Typewriter: The Evaluation of a Prototype," *5th International Seminar on Rehabilitation.* London, England, 1974, pp. 1-21.
31. Vanderheiden, Gregg Charles, Gerald A. Raitzer, David P. Kelso and C. Daniel Geisler, "An Automated Technique for the Interpretation of Erratic Pointing Motions of Severely Cerebral Palsied Individuals," submitted for publication, 1975.
32. Vanderheiden, Gregg Charles, Gerald A. Raitzer, David P. Kelso and C. Daniel Geisler, "A Portable Non-Vocal Communication Prosthesis for the Severely Physically Handicapped," submitted for publication, 1975.
33. Vanderheiden, Gregg C., Andrew M. Volk and C. Daniel Geisler, "An Alternate Interface to Computers for the Physically Handicapped," *AFIPS Conference Proceedings of the 1974 National Computer Conference,* Vol. 43, pp. 115-121.
34. Vanderheiden, Gregg C. and Mary Jo Luster, *Non-Vocal Communication Techniques and Aids as Aids to the Education of the Severely Physically Handicapped: A State of the Art Review, 1975,* (In preparation) Trace Center, University of Wisconsin-Madison, Madison, Wisconsin 53706.
35. Ziskind, A. and R. Ziskind, "Remote Control Typewriter for Paraplegics," *Journal of American Medical Association,* 169, January 3, 1959, pp. 459-460.

APPENDIX

AID REFERENCES

Alphabet-Message Scanner
Prentke Romich Company
R.D. #2, Box 191
Shreve, Ohio 44676
(215) 567-2906

Auto-Com
Trace Center
922 ERB, 1500 Johnson Drive
Madison, Wisconsin 53706

Clock Face Selector (C.F.S.)
Centre Industries
Allambie Road
Allambie Heights
New South Wales,
Australia

Code Operated Selector (C.O.S.)
Centre Industries
Allambie Road
Allambie Heights
New South Wales,
Australia

Comhandi
Physico-Medical Systems Company
9250 Park Avenue, Suite M-101
Montreal 354, Quebec
Canada

Communications Prosthesis for the Cerebral Palsied
Loren J. Wymore
3 Gregory Court
Banington, Rhode Island 02806

Computer Assisted Communication (CAC)
United Cerebral Palsy of Middlesex Co.
Roosevelt Park
Edison, New Jersey
or
The Telephone Pioneers
Bell Telephone Laboratories
Whippany Council
Whippany, New Jersey

Corsford Selector
Mr. Ellis Cohen
19 Cochrane Street
Glasgow, C.I.
England

Discom
Rikscentralen
Bracke Ostergard
S-417 22
Goteburg
Sweden

Electronic Typewriter Controller
Kingma Harding Associates, Ltd.
9639-62 Avenue
Edmonton, Alberta TGE 0E1
Canada

Enlarged Typewriter Keyboard (adapted IBM Selectric Typewriter)
Suzanne D. Hill
Department of Psychology
University of New Orleans
Lakefront
New Orleans, Louisiana 70122

Eye controlled "Teletypewriter"
Technology Utilization and Application Programs Officer
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Herald I & II Electronic Communication Boards
The HERALD Company
3840 Railroad Avenue
Pittsberg, California 94565

Keyboard for Operation with the Feet
MEFA GmbH Bonn
518 Eschweiler
Post Sach 466
Germany

Lightspot Operated Typewriter (LOT)
Dr. H. G. Stassen
Associate Professor of Control Engineering
Lab for Measurement and Control
C. Drebbelweg 1
Delft University of Technology
Delft
The Netherlands

Linquaduc
Carba, Inc.
Technischer Berater
3097 Liebefeld-Bern
Switzerland

MC 6400 Communicator
MEDICEL
222 Foxbill Road
Burlington, Massachusetts

MCM—Manual Communications Module
Silent Communications, Inc.
1440 29th Avenue
Oakland, California 94601

Multi-Access Interface for the Disabled (MAID)
Mr. J. Agzarian
Department of Medical Physics
The Prince Henry Hospital
P.O. Box 233
Matraville, New South Wales 2036
Australia

Modified Electric Typewriter
The National Institute for Rehab. Engineering
Pompton Lakes, New Jersey 07442

OCCUR—(Optical Controlled Communication Unit for Rehabilitation)
National Research Council
Ottawa, Canada
K1A0R8

PILOT System (Patient Initiated Light Operated Telecontrol)
Hugh Steeper (Rechampton) Limited
Queen Mary's Hospital
Rochampton Lane
London, S.W.
England

PMV Keyboards
PMV Printer
PMV AB
Dobelnsgatan 34C
113 52 Stockholm
Sweden

POSSUM Typewriter Control Systems
Possum Controls, Ltd.
63 Mandeville Road
Aylesbury
Buckingham HP21-8AE
England

Rumble Communicator MK 1
L. A. Rumble
30 Benton Road
Ilford, Essex, I6I 4AT
Great Britain

System 8
Zambette Electronics
17 High Street
Southend-on-Sea
Essex
England

Tongue Controlled Typewriter
Technical Aids to Independence
12 Hyde Road
Bloomfield, New Jersey 07003

TV Phone
Phonics Corporation
814 Thayer Avenue
Silver Spring, Maryland 20910

Whispertype
Cybernetics Research Institute
2233 Wisconsin Ave N.W.
Washington, D.C. 20007

The spellex system of speech aids for the blind in computer applications

by CHING Y. SUEN
Concordia University
Montreal, Canada

MICHAEL P. BEDDOES
University of British Columbia
Vancouver, Canada

and

JAMES C. SWAIL
National Research Council
Ottawa, Canada

ABSTRACT

Spellex is a system developed to aid the blind in the use of computer and office equipment without sighted help. It consists of a digital spelled speech generator interfaced with a number of instruments by digital electronic circuits. Voice is produced by two specially programmed read only memory chips. Standard C-MOS components are used in the design of the spelled speech generator so that the processor can be easily implemented using the latest large scale integrated circuit technology. As it now stands, Spellex provides voice for standard input and output terminals, punch card reader, paper-tape punch and reader, calculator, electric typewriter and the Lexiphone reading machine for the blind.

INTRODUCTION

With the rapid progress in digital electronics and speech processing techniques, the field of aids for the blind has entered the stage of synthesizing voice by digital circuits as the output medium.[1, 3] This evolution of using spoken sound as output is logical because speech is one of the most common and natural media of human communication. Due to the decrease in price of electronic components, it is now possible to build intermediate types of speech aids at a reasonable cost. This paper deals with the application of the Spellex system for communication with the computer and its accessories.

Many job opportunities in our competitive society are not available to the blind because they cannot work without sighted help. However, with a multitude of aids now being developed using tactile and auditory feedbacks,[5] it is forecast that more blind people will be able to hold technical and professional jobs in the future. Indeed, during the past several years a considerable number of blind programmers were able to find employment in the field of data processing. There are also several good facilities for the training of blind programmers, both in North America and Europe.

In response to a questionnaire on their special needs, blind programmers indicated that they would like to see the following tools developed:[8] Braille terminals and printouts, reader for inkprint printouts, spoken input/output, card reader for verifications and corrections and Braille writer as an attachment to miscellaneous equipment. The development of the Braille line printer[4,6] has answered some of their needs and the development of the Spellex system will further resolve their problems related to reading and communication with the computer and its accessories.

The Spellex system depicted in Figure 1 is the result of several years of continuous research and development in auditory feedback. At the present stage of development, it can generate a synthesized spoken sound each time it receives an output signal in ASCII from the various machines. The spelled speech generator was initially conceived as an output device for the Lexiphone reading machine.[7,8] This enables a blind user to monitor the machines easily. At the present moment, the Lexiphone can read several type fonts.[2] A calculator has been interfaced to Spellex and an IBM electric typewriter is functional with an editor and enunciates the sound of all the keys. The system also

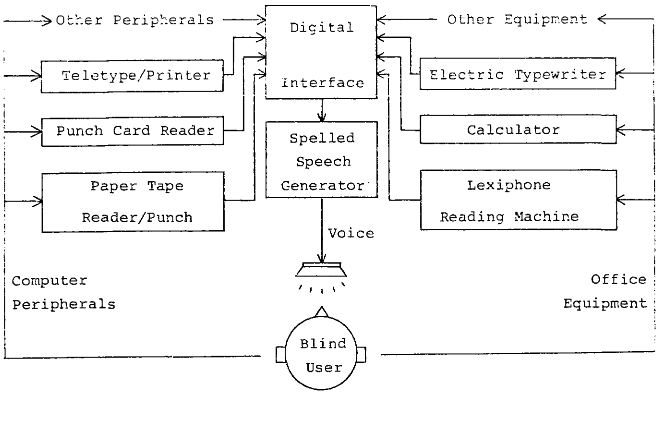


Figure 1—The Spellex system

operates a portable punch-card reader, teletypes and teleprinters.

THE SPELLED SPEECH GENERATOR

Spelled speech was synthesized by the concatenation of phonemes stored digitally in Read Only Memories (ROMs). In order to minimize the memory requirement, only 18 phonemes were chosen to synthesize the whole set of 64 ASCII characters.[7] They were extracted from recordings by the computer specially programmed for digitization and segmentation purposes. Although some of the sounds produced may be a bit artificial, extensive experiments proved that blind subjects could master them in a matter of just an hour or so after which they could "read" between 60 and 80 words a minute.[8]

The first version of the generator was developed on a mini-computer. Based on the principles of the computer program which generates the spoken sounds, a portable digital apparatus was subsequently designed and built. The circuit is relatively simple and can be easily converted to the current LSI computer technology. Its block diagram is shown in Figure 2. As the ASCII code of a character reaches the machine, a pointer is set to search its location in memory. By a table look-up technique, the phoneme combination for the corresponding sound is found. The phonemes are then generated through the digital to analog converter according to pre-programmed phonetic rules also stored in the ROMs. Thus, one hears the sound A (phonemes /e/ and /i/) when the ASCII code 301 reaches the machine. The current spelled speech generator will make any instruments talk as long as they give an ASCII output.

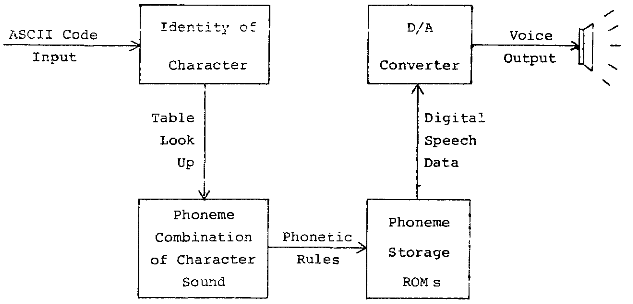


Figure 2—Block diagram of the spelled speech generator

COMMUNICATIONS WITH THE COMPUTER

Since ASCII codes are readily available from most computer terminals, the Spellex system enables blind programmers to hear what they command the computer to do. This mode of operation is desirable for mini-computers as well as big computers using remote terminals. Our system provides communication with the computer by telephone lines at remote terminals at three locations in Vancouver: the Jericho Hill School for the Blind, the Crane Memorial Library and the Vancouver branch of the Canadian National Institute for the Blind. An editor is also included in the programs so that detected errors can be removed immediately. Once the user is satisfied with the output, he can have it punched on paper-tape so that multiple copies can be made afterwards. So far, blind students have used the system for text editing and as a training facility for typing. They have used it to type theses, make copies, circulars, and even applications for jobs! Nearly everyone likes to play with the new machines. However, continued co-operation with blind people is needed to adapt the interface to new uses as they are discovered. Many suggestions have been received from users of these terminals, especially from the oldest installation (1972) at the Jericho Hill School for the Blind. The system has also been used by multiple-handicapped blind.

So far, we have answered the input problem. The output problem is more difficult to solve due to the fast speed at which characters are generated from the computer. Even with the slowest specified rate of 110 bauds, it represents an output speed of at least 100 words a minute, and the speed of line printers could be many times faster. Some blind people can listen to spelled speech at 100 words per minute if they have some ideas about the kind of output and the likely results, but not the others. To overcome this particular problem, we have programmed the computer to slow down its speed by the introduction of silent gaps between characters. To date, we have used this technique with success. Another alternative is to implement a memory buffer at the terminal so that the output can be slowed down to the programmer's desired speed. Of course one can also record the computer output on tape-recorders so that he can verify it when desired. The reading of paper-tapes follows the above pattern.

CARD READING

The one operation with which a blind programmer does not appear to be able to cope satisfactorily is the reading of individual punched cards for verification and correction. Braille printouts may be used to check a complete stack of cards, but there is a need to read cards singly for certain purposes. One device, which has been available for some time, requires the user to find each hole with a probe, but the procedure is too time-consuming for efficient operation.

To answer the above needs, a portable card reader was developed in the laboratories of the National Research Council. This device employed a tactile read-out with no form of code conversion. Appropriate tactile stimulators in a row of 12 were activated corresponding to the holes punched on the card. Recently we have modified this equipment to provide an ASCII output for the Spellex spelled speech generator. A block diagram of this new device is shown in Figure 3.

The card to be read is placed in a carriage that is movable from left to right. A small plastic key in the rear left hand corner assures correct orientation of the card. Only one column is read at a time and this is selected by moving the carriage manually. Column positions are identified by feeling the location of a pointer which moves along a raised scale. A row of twelve photocells under the moving carriage is fixed to the body of the instrument to detect light passing through the 12 possible punched holes in any column. A fluorescent lamp is used as the light source. ASCII codes of the punched characters are provided by an integrated circuit Hollerith to serial ASCII code converter. Thus, the operator hears the sound of the punched character each time its holes pass through the photocells. At present, the synthesizer is triggered at each column location by a pushbutton. However, it is planned to provide an automatic trigger so that each character on the card will be pronounced as soon as the column is properly aligned. An automatic drive for the carriage will also be added if field trials with blind programmers indicate that this is a useful feature.

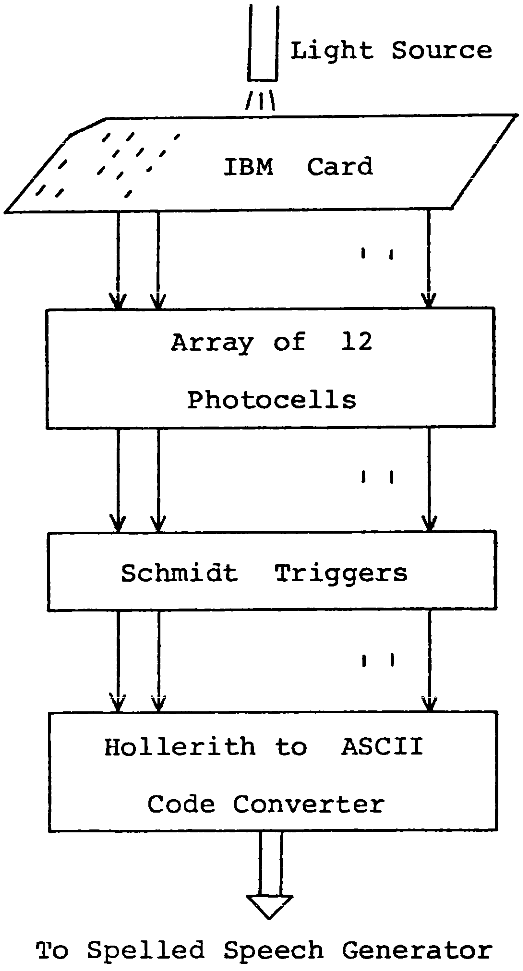


Figure 3—Punch card reader

CONCLUSION

The Spellex system does not only make office equipment such as the typewriters and calculators talk but also teletypes, card readers and some other computer peripherals. In sum, it can make any instruments talk as long as ASCII code is available. Although spelled speech is slower in speed than ordinary conversation, it is especially useful in such cases as reading computer programs, checking spelling errors and providing numerical output. It can also be employed in the mini-computer environment, on-line applications and satellite terminals for large computers. It has the potential of being manufactured by large scale integration technology. The cost of such a unit can be very low when large quantities are produced.

OUTLOOK FOR THE FUTURE

Over the past few years, numerous blind people have accepted the challenge of working in the field of data processing. While scientists and research workers have developed a number of tactile and auditory instruments to assist them, there are still some problems blind programmers are facing, in particular: faster means of reading computer output, flow-charting communication, means of getting information from books, manuals, periodicals, and other reference materials. It seems that the needs of blind programmers are many and can only be solved when a multi-media approach is taken, i.e., combining the use of both tactile and auditory feedbacks for adaptation to computer peripherals and other equipment. They could then use the appropriate type of sensory aid as required. Those who are poor in Braille and tactile senses (e.g., diabetes) could make use of auditory instruments to carry out their daily work. The converse is true for those who have a poorer auditory sense although statistically this condition is far less common than the former condition. While Braille seems to be more suitable for establishing permanent records of computer output and programs, spelled speech is more suitable for the input phase and on-line monitoring, and optical readers such as the Optacon and the Lexiphone are more suitable for reading materials already in print. A multi-media approach in a time-sharing mode seems ideal for the visually handicapped to choose the right equipment and feedback medium for each particular situation. The ultimate solution will be a low-cost multi-media system which can only be realized by greater efforts in research, development and evaluation.

ACKNOWLEDGMENTS

This work was supported by grants from the Medical Research Council and the National Research Council of Canada, the Mr. and Mrs. P. A. Woodward Foundation, the Vancouver Foundation, and the Department of Education of Quebec, Canada. The authors also wish to thank John Moon and Lloyd McSheffrey for assistance.

REFERENCES

1. Allen, J., "Reading Machines for the Blind: The Technical Problems and the Methods Adopted for Their Solution," *IEEE Trans. Audio and Electroacoustics, AU-21*, pp. 259-264, 1973.
2. Beddoes, M. P. and C. Y. Suen, "Transducers for a Reading Machine for the Blind," *Proc. International Conference on Biomedical Transducers, 2*, pp. 511-515, Paris, Nov. 1975.
3. Cooper, F. S., J. H. Gaitenby, I. G. Mattingly and N. Umeda, "Reading Aids for the Blind: A Special Case of Machine-to-man Communication," *IEEE Trans. Audio and Electroacoustics, AU-17*, pp. 266-270, 1969.
4. Dalrymple, G., "Sensory Aids Progress at the MIT Sensory Aids Evaluation and Development Center," *Research Bulletin of the American Foundation for the Blind, 27*, pp. 11-44, 1974.
5. Nye, P. W. and J. C. Bliss, "Sensory Aids for the Blind: A Challenging Problem with Lessons for the Future," *Proc. IEEE, 58*, pp. 1878-1898, 1970.
6. *Proceedings of the First International Workshop on Computerized Braille Production*, edited by R. A. J. Gildea, G. Hübner and H. Werner, ACM SIGCAPH Newsletter, March 1975.
7. Suen, C. Y. and M. P. Beddoes, "Development of a Digital Spelled-Speech Reading Machine for the Blind," *IEEE Trans. Bio-Medical Engineering, BME-20*, pp. 452-459, 1973.
8. Suen, C. Y. and M. P. Beddoes, "Spelled Speech as an Audio Output for the Lexiphone Reading Machine and the Spellex Talking Typewriter for the Blind," *Research Bulletin of the American Foundation for the Blind, 29*, pp. 51-66, 1975.
9. *Summary of Responses to Blind Programmer Questionnaire*, ACM SIGCAPH Newsletter, pp. 5-13, July 1973.

Development of a hand-held talking calculator for the blind

by R. E. SAVOIE, J. S. BRUGLER and J. C. BLISS
Telesensory Systems, Inc.
Palo Alto, California

ABSTRACT

This paper describes a project to develop a calculator for blind people. Based on the results of a survey of needs and an evaluation of various types of displays for the blind, spoken word output was chosen as the ideal display. The final product, hand-held and battery powered, incorporates a custom LSI microcontroller and a single 16K Read Only Memory to synthesize a vocabulary of 24 words.

INTRODUCTION

Small, lightweight hand-held calculators have become as standard a household item as toasters since Hewlett-Packard Company pioneered the HP-35. These electronic calculators are among the few truly innovative products of recent years and a success of American technology made possible by Large-Scale-Integration (LSI) electronic circuitry. New and improved calculators are being introduced almost daily by various manufacturers, with computation capability ranging from the basic four functions to programmable models which are essentially computers. All indications are that the prevalence of use of these calculators by working adults and school children is on the increase.

Thus far, the blind have largely not benefited from this technological advance because pocket calculator displays are visual. To overcome this lack, many organizations, including our company, have worked at developing special calculators suitable for use by the blind. The motivation for these efforts stems largely from the realization that an effective calculator for the blind would not only provide a quick and reliable means for doing calculation, but also would offer the possibility of increasing the mathematical skills of the blind. Mathematics has traditionally been a difficult subject for blind students, partly because of the complications of displaying mathematics in Braille.

The development described in this paper initially addressed several fundamental questions such as: How would the blind use a calculator? What features are required? How should the information be displayed? The answers to these and other questions led us to develop a talking calculator that provides to the blind user all the information and convenience that ordinary hand-held calculators provide to the sighted.

BACKGROUND

The traditional calculation tool for the blind has been the abacus, which performs simple computations adequately, but which requires great skill for more involved work. When electronic calculators became generally available, TSI developed a special lens module that allows an Optacon user to read many of these calculators without modification. The Optacon is an electronic reading aid for the blind that presents the user with a tactile facsimile image of an area about the size of a letterspace, selected by a small camera that is scanned by the user. By replacing the standard lens module used for reading inkprint with the special calculator lens module, the Optacon user can scan the calculator's visual display and feel the images of the digits displayed on his finger. A number of Optacon users own sophisticated, scientific calculators which they use routinely in their work. However, many totally blind or low vision people are not Optacon users, but have a need for a portable calculator which can produce the numerical display in a form that does not have to be read visually. To address these needs, TSI began to investigate the alternatives to the visual display readout which is standard on pocket calculators.

DESIRED FEATURES OF A CALCULATOR FOR THE BLIND

Blind persons' computational needs cover the same wide spectrum as that of the general public. In order to accurately assess interest from the blind community in an electronic calculator, we conducted an informal survey of 180 blind people throughout the country.[1] We found that 75 percent of those questioned expressed a desire to own an electronic calculator. The results of the survey, together with our own opinion, led to the following list of key important features needed in any calculator we should develop for the blind:

1. *At least four functions (add, subtract, multiply, and divide), floating point calculation plus automatic constant, memory and square root.* Nearly half of the survey respondents indicated that they would use a calculator primarily for personal use, and only 5 percent needed full scientific capability. The above features would therefore cover most of the expected usage.
2. *Easy usage by the sighted and partially sighted.* Many potential blind users have sighted spouses and children who would also use the calculator. The utility of a calculator would also be increased considerably if those partially sighted who cannot read ordinary calculators would be able to use it.
3. *Easy portability and battery operation.* About half of our survey respondents indicated a firm desire to have a portable unit as close as possible to pocket size.
4. *Immediate confirmation of keyboard entry.* The operator needs instant feedback to confirm each key-press to insure easy learning and fast, accurate computation.
5. *A display that is easily learned.* To enable widespread usage there should be no undue burden on the operator, other than learning how to operate the calculator. Additionally, the calculator should be easy to teach and demonstrate.
6. *Minimum human error in display readout.* Calculators do not make mistakes—human operators do. The display technique chosen should be sufficiently clear and unambiguous to minimize the inevitable human error.
7. *Low "user frustration."* This subjective parameter relates to the others, and means basically that the display is easy to use. The operator should be able to concentrate on the mathematical problem at hand, not on just reading the display.

A critical problem in meeting the above goals is to choose the proper display. The computation itself is easy, due to the recent availability of inexpensive LSI calculator circuits. Our initial work was therefore directed toward the display problem, with the hope that the same LSI technology could lead us to an effective display. Since we were searching for "pocket" calculator displays, we did not consider displays providing permanent hard copy.

DISPLAY ALTERNATIVES AND EVALUATION

Choice of a digital display for the blind is the subject of considerable effort throughout the world. A recent report[2] by J. M. Gill of the University of Warwick, Coventry, England lists 27 different digital displays either available or proposed. We encountered several additional displays during our investigations. Most of these displays, it should be noted, are useful for digital information whether from a calculator or not.

In Table I we present a listing of the important pocket calculator displays that we know, along with a comparison with respect to the important features we identified in our survey, plus calculation speed and cost. Displays are ranked within each category, with lowest numbers the best. To the right, scores of each column are added to give a rough ranking of the various display options. Our listing of the relative display merits is, of course, subject to debate, but the indication is that the most attractive display is speech.

One can appreciate that the display problem is really one of too many choices. All of these displays work. Undoubtedly, any one of them could be happily and successfully used by many people. Faced with this dilemma, we decided to construct or obtain several different calculators having the display techniques we liked the best, and conduct a comparative evaluation as objectively as possible. Following this plan, we evaluated four different calculators.

1. *An Audio-Matrix Scan Display*—a number of side-by-side columns of braille readout cells, one column for each display digit. The user scans each column of cells vertically with his fingertip. A tone is provided when the correct cell is touched.
2. *An Audio-Keyboard Scan Display*—the calculator keyboard itself is used to perform the digital readout. The user scans the keyboard by sequentially pressing the numeric keys. An audio tone is generated when the key corresponding to the

TABLE 1—Ranking of Calculator Displays

Display Type	Learning Effort Required	Sighted Use	Computation Throughput Speed	Portability	Confirmation on Entry	Overall Error Rate	Cost	User Frustration	Unweighted Total Score
Abacus	2	2	4	1	4	2	1	2	18
Optacon	5	1	1	2	3	2	5	2	21
Tactile-Parallel	3	3	1	2	2	2	4	2	19
Tactile-Serial	3	3	2	1	2	2	3	3	19
Audio Tones	4	3	2	1	3	4	2	3	22
Speech	1	1	2	1	1	1	4	1	12
Audio-Keyboard Scan	2	2	3	1	2	3	2	3	18
Audio-Matrix Scan	2	2	3	2	2	3	2	3	19

most significant digit is depressed. At successive scans, successive digits are determined on down to the least significant digit. Of course, the calculator itself is disabled during the scanning process.

3. *A Braille Serial Display*—a single braille display cell dynamically presents the digits, one at a time, from most to least significant digit upon user command.
4. *An Audio-Speech Display*—synthetic speech is the output code.

Our evaluation involved participation by almost two dozen blind subjects, many of whom were sophisticated in mathematics and had experience in evaluating sensory aids. The evaluators were shown the calculators and allowed to practice. Next, they were timed for speed and accuracy doing sample problems. Finally, their subjective preferences were solicited.

The Audio-Keyboard Display was eliminated early in the evaluation because we determined that it simply required too much effort on the part of the user. Of the seven evaluators who saw the remaining three displays, six preferred speech output, and speech proved to be the fastest and most accurate, as shown below.

Display Type	Av. Time/ Problem	Total No. Errors	Preferred by
Matrix Scan	66 sec.	13	0
Braille Serial	38 sec.	14	1
Audio Speech	26 sec.	0	6

A key finding was that quite commonly an error with the matrix or braille displays was the result of reading correctly a display that had the incorrect answer because the user had made a mistake entering the problem without knowing it. This problem was obviated with the speech display, which pronounced the name of each key as it was depressed.

Our evaluation thus led us to conclude that spoken speech is preferred using both subjective and objective measures. Speech, as the natural human communication means, uniquely needs no learning, can be used by the sighted, causes minimum user error and frustration, and provides immediate feedback identifying every key depression. Another advantage of voice for the calculator application is that additional required display parameters, such as "overflow," "minus" and "point" are provided in a natural way. All of the other display techniques become awkward when faced with these extra functions.

DEVELOPMENT OF SPEECH PLUS™ CALCULATOR

Based on the results of our survey and evaluations, we formulated the following product specifications:

1. Hand held and battery operated
2. Spoken speech for every key depression and for display on command
4. Cost effective and reliable.

One main effect of these constraints was to eliminate analog methods of producing speech, such as magnetic tape or optical film, which we felt could not compete with modern solid-state electronics in terms of size, cost and reliability.

At the time that these goals were written, we had a breadboard talking calculator (used in the evaluation) that comprised two large boxes full of electronics. One box contained a TTL prototype electronic speech synthesizer which employed a synthesis algorithm for which TSI subsequently obtained an exclusive license. The remaining electronics was required primarily to interface our speech synthesizer to an ordinary calculator chip. This interface proved to be surprisingly complex, because the high-speed, multiplexed, 7-segment signals used for the visual display are inappropriate for the slow-speed sequential readout required by the speech synthesizer. A possible solution to eliminating this interface would have been the use of a printing calculator chip, but there were none available that satisfied our computational needs. The major engineering tasks as we began the development can be summarized as:

1. Reduce the speech synthesizer (over 60 integrated circuits) to a small, cost effective size
2. Reduce the complexity of (or eliminate entirely) the interface between the calculator chip and the speech synthesizer
3. Design a convenient package engineered with the human factors of a blind user in mind.

To tackle the problem of the speech synthesizer, we engaged a consulting firm to evaluate the synthesis algorithm and recommend whether it could be implemented using a microcomputer. Their findings supported our feelings that microprocessors available then (April 1975) were inadequate in all of the following areas:

1. Speed
2. Size
3. Power consumption
4. Cost

We had previously confirmed the feasibility of executing the synthesizer design in a custom LSI microcontroller. With the finding that the microprocessor approach was not feasible, we began LSI development. The resulting custom MOS LSI program successfully reduced the 60 IC packages to a single microcontroller circuit and a single ROM.

The microprocessor approach, however, was the solution to reducing the interface complexity. We were able to identify a one-chip microcomputer that could be programmed to perform the computational algorithm we required and also to give the speech command signals required by the speech synthesizer.

The packaging problem involved innumerable details, but was centrally concerned with satisfying the following requirements:

1. Small and lightweight enough to be hand-held
2. Keyboard designed expressly for the blind.

The first problem was complicated by the presence of considerably more electronics than are in an ordinary calculator. The major contribution to size reduction was, of course, the use of LSI integrated circuits to reduce the package count. We also took care to assure low power consumption so we could obtain the required operating time with physically small batteries.

To find out as much as we could about the best keyboard design, we conducted an evaluation of twenty commercial keyboards using almost fifty blind subjects. We also tested several different formats of custom keyboard. For these, subjects were timed and scored for errors on sample problems with a simulation of a talking calculator. The general findings, which were incorporated into the calculator design, were:

1. Relatively large key tops were preferred
2. Relatively long keystroke was preferred
3. A tactile locating "bump" on the "5" key was preferred
4. Functional partitioning by spatial separation was more efficient than other tactile cues
5. A 6×4 horizontal key matrix was more efficient than a vertical or square format (because many subjects used two hands to input data)
6. The standard calculator arrangement of the number keys was disliked by many blind subjects. These subjects had experience using push-button telephones, which have an inverted arrangement. It would be confusing to a blind user to alternate between calculating and telephoning if the basic number matrices were different. We therefore decided to use the telephone arrangement for the number matrix of the keyboard

Figure 1 shows a functional block diagram of the talking calculator. It employs a "three-chip" architecture. On the left is the microcomputer, which implements our calculation algorithm, drives the visual LED display, services the keyboard, and sends speech command signals to the speech synthesizer portion on the right. The speech command signal is a 5-bit parallel code specifying one of the 24 words in the calculator's vocabulary, and a strobe signal to initiate speech.

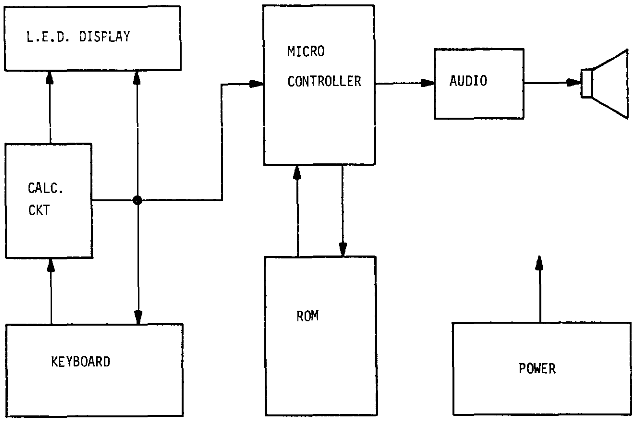


Figure 1—Functional block diagram.

The speech synthesizer uses two LSI integrated circuits, one of which is a commercially available 16-K n-channel MOS read-only-memory, mask programmed with our custom pattern. The second IC is our Custom MOS ROM Controller (CRC), a micro-controller which implements a proprietary algorithm for speech synthesis. Upon command of the calculator chip, the micro-control chip reads the outputs of the calculator chip. These signals are used to fetch control information stored in the read-only-memory (ROM) chip. The CRC also determines the addresses needed to access data stored in the ROM. From ROM information, the control chip determines how to say the word, the pronunciation, how long to say the word, and when to stop saying the word.

Sound is produced from data stored in the ROM. The micro-control chip selects the ROM addresses to be read and speech is produced from the data stored in a given location. Each speech sound is made up of many digital bits, each one making up an increment of the analog audio signal. The control chip also converts the digital information produced in conjunction with the ROM to an analog audio signal, via an on-chip D/A converter.

The division of labor between these two ICs is such that the vocabulary and language information reside solely within the ROM, while the controller is language- and vocabulary-independent. Thus, a change in the language of the calculator involves only replacing the "English" ROM with the appropriate foreign language. We are presently working on a German calculator.

APPLICATIONS

The Speech Plus™ calculator is expected to open whole new fields of application of math to the blind. We see imoprtant uses in the vocational, educational, and home areas.

Blind vending stand operators can use the calculator to figure prices and sales taxes, and also to do their own accounting. Blind salesmen can figure quotes, budgets, and the like. And scientists and engineers can use the calculator at their jobs.

In education, the calculator offers the possibility of improving the methods of teaching math to blind (and sighted) children, as the entire class can follow the calculation by sound. At higher grades, it offers a means for blind high school and college students to do assignments involving calculations.

In the home, the calculator can be used to reconcile bank accounts, figure taxes, do comparison shopping, and is useful in hobbies such as carpentry and electronics.

We intentionally structured the calculator so that the calculating portion and the speaking portion are relatively independent. The speech synthesizer can easily be expanded to a 64-word vocabulary with the addition of one more ROM. There are many applications for other devices for the handicapped which could benefit from the use of such a relatively small-vocabulary voice capability. These include talking voltmeters, talking clocks, talking typewriters, talking notetaking devices, talking computer terminals, and so on. The development of a speaking capability for the calculator thus opens the door to a wide variety of other possible applications.

SUMMARY

An extensive survey of blind people indicated that there is demand for a hand-held electronic calculator useable by the blind. An evaluation with blind subjects showed that audible speech is the best display modality for an electronic calculator for the blind. The principal drawback of speech is its cost, which has been effectively mitigated through LSI electronics.

REFERENCES

1. Proscia, Vito A., Robert E. Savoie and J. S. Brugler, "Electronic Calculators for the Blind," presented Krusen Center for Research and Engineering at Moss Rehabilitation Hospital, Temple University Health Sciences Center Program, *Devices and Systems for the Disabled*. April 29 and 30, 1975.
2. Gill, J. M., "Non-Visual Computer Peripherals," survey report distributed by Department of Engineering, University of Warwick. September 1974.

Survey of public attitudes toward computers in society*

by DAVID H. AHL
AT&T and Publisher—Creative Computing Magazine
Morristown, New Jersey

ABSTRACT

There is no doubt that the computer will have at least as great an effect on humankind as any previous innovation, but today we can see just the tip of the iceberg. In the future the computer will be not merely in the realm of the scientist or data processing specialist but it will be available to everybody.

Do people understand the computer and what it can do? *Creative Computing* magazine conducted a survey in 1975 to find out. The survey was conducted in two highly computerized nations, the United States and Germany among both young people and adults and among people who had been exposed to the computer and those who had not. The sample of 843 was relatively balanced demographically.

This survey indicates that most people are remarkably optimistic about the benefits the computer can bring to society in a number of areas, for example, education, law enforcement, and health care. People feel they are unable to escape the influence of the computer and that it has some undesirable effects, however, they do not feel particularly threatened by it. Young people tend to be less optimistic and feel more threatened by the computer than do adults. A surprising two-thirds of the population have a fair understanding of both the role and function of the computer although there are a few misconceptions.

Compared to the 1971 AFIPS/*Time* survey, people have become more optimistic about the use of computers in most areas with the notable exception of credit data banks. Also, this *Creative Computing* survey identified the computer influence on elections as a real danger area—to our knowledge this has not been previously surveyed.

METHODOLOGY

During the 6-month period, February through July 1975, *Creative Computing* Magazine conducted a survey on people's attitudes toward computers and their role in society. Some 843 people responded in two highly computerized nations, the United States and Germany. About one-third of the respondents had been directly exposed to computers in some way; two-thirds had not. Thirty-six percent of the respondents were classified as young people (20 and under) and students; the remainder were a relatively balanced cross-section of adults.

The 17 questions in the survey fell in four major categories (although they appeared in random order on the survey instrument). The categories:

1. Computer Impact on the Quality of Life (4Q)
2. Computer Threat to Society (4Q)
3. Understanding of the Role of Computers (5Q)
4. Understanding of the Computer Itself (4Q)

In some cases where the questions were similarly worded, the responses to this questionnaire are compared to those from a 1971 survey jointly sponsored by AFIPS and *Time* Magazine.

SUMMARY

Computers are not only invading our lives along a multitude of directions—supermarkets, credit data, medical records, hobbies, etc.—but our society is becoming so dependent upon computers that it can truly be said that we live in the computer age. The computer will have at least as profound an effect on humankind as did the printing press some 500 years ago. In the Guttenburg Museum, a map plots the spread of printing out from Mainz to the rest of the world over scores of years. The computer invasion has taken place at an infinitely greater speed.

Now, some 30 years after its invention, what do people think of the computer. Monster or savior? Slave or dictator? Do people understand this awesome force?

This survey indicates that most people are remarkably optimistic about the benefits the computer can bring to society in a number of areas, for example, education, law enforcement, and health care. People feel they are unable to escape the influence of the computer and that it has some undesirable effects, however,

* For further information about this survey, please contact David Ahl, Editor, *Creative Computing*, P.O. Box 789-M, Morristown, N.J. 07960.

they do not feel particularly threatened by it. Young people tend to be less optimistic and feel more threatened by the computer than do adults. A surprising two-thirds of the population have a fair understanding of both the role and function of the computer although there are a few popular misconceptions.

Compared to the 1971 AFIPS/*Time* survey, people have become more optimistic about the use of computers in most areas with the notable exception of credit data banks. Also, this *Creative Computing* survey identified the computer influence on elections as a real danger area—to our knowledge this has not been previously surveyed.

COMPUTER IMPACT ON THE QUALITY OF LIFE

On the whole, respondents felt that the computer will improve the quality of life in four areas: education, law enforcement, health care, and prevention of fraud. Young people and students saw somewhat less improvement from the use of computers than did adults.

Computers will improve education—About 85% of all the respondents strongly or mostly agreed with that statement and only 5% disagreed. This was the highest positive (or negative) response to any single question and also the question which had the greatest agreement between adult and youth.

Computers will improve law enforcement—82% of the adults agreed with this and only 3% disagreed. The younger respondents were somewhat more cynical; 70% agreed and 10% disagreed.

Computers will improve health care.—On this issue, the young respondents had considerably more doubts than adults; about 79% of the adults agreed but only 54% of the youth. More than twice as many youth disagreed with the statement as adults—12% vs 5%.

Ranking lower on desirable uses of the computer is its use for *storing and checking credit rating data;* 64% of both adult and youthful respondents saw this as a worthwhile application. However, 13% of the adults thought this was a bad application for the computer perhaps reflecting previous hassles that they or friends had with computerized credit rating data. Most young people probably haven't been exposed to this malady; only 8% of them objected to this application. While substantial, the 64% of the people in favor of this application represents a substantial decline from the 75% recorded just four years ago in the AFIPS/*Time* survey.

THE COMPUTER THREAT TO SOCIETY

Respondents were mixed in their feelings about the threatening nature of computers. Most felt they were unable to escape the influence of the computer. Nearly half saw computer predictions influencing the outcome of elections. More than one-third felt that computers dehumanize society to some extent. About one-quarter saw the computer taking more jobs than it creates. And about one-fifth saw the computer having an isolating effect on programmers, operators, etc.

A person cannot escape the influence of computers—92% of the adults agreed with that statement, most "strongly" agreeing; only 4% disagreed. These percentages are virtually the same as those recorded in the 1971 AFIPS/*Time* survey. Reflecting a more optimistic, perhaps somewhat naive view that one can drop out and avoid anything one doesn't approve of, only 67% of the young people felt they could not escape the influence of computers and 18% strongly disagreed with the notion that computers couldn't be avoided.

Computer polls and predictions influence the outcome of elections.—About 46% of the respondents agreed with this statement and 27% disagreed. In a democratic society, this is truly of grave importance. If almost half the people, adult and youth alike, feel their voting behavior is in some way influenced by computer polls and predictions (join the bandwagon, we've lost so why bother voting, etc.) then we have a real problem.

Computers dehumanize society by treating everyone as a number.—Reflecting a rather positive shift in attitude, only 37% of the adult respondents agreed with this statement and 50% disagreed compared to the percentages of 54% agreement and 40% disagreement just four years ago in the AFIPS/*Time* survey. The younger respondents in the current *Creative Computing* survey were more pessimistic; 40% agreed that computers dehumanize and only 31% disagreed. (Youth were not included in the 1971 survey.)

Computers isolate people by preventing normal social interactions among users.—Computer bums and computer freaks are common around any school with a computer. Million dollar corporate computers have to be fed data around the clock to justify their investment and there is a growing army of midnight shift programmers and operators. Among the uninitiated, FORTRAN or COBOL are more foreign than French or German. Are computers really isolating segments of society? Maybe, but apparently it's not very noticeable since only 20% of the respondents agreed with the statement above. More revealing, however, is the fact that 63% of adults disagreed with the statement and only 43% of young people disagreed. Perhaps computer freaks, who tend to be among the younger cadre, *are* becoming more evident.

UNDERSTANDING THE ROLE OF COMPUTERS

This issue was examined from two directions: what types of jobs are suitable for a computer and what will be its effect on human employment (or unemployment)? For the most part, adults saw the computer as suitable for dull, repetitive tasks like a hammer or lathe while young people saw computers in much broader roles. Furthermore, adults saw computers replacing low skill jobs and creating just as many jobs as they eliminate; young people were not as optimistic.

Computers are best suited for doing repetitive, monotonous tasks.—Eighty percent of the adults agreed with this statement and 10% disagreed. Among young people, 57% agreed, 22% disagreed. In other words, young people see the computer doing a wide variety of things beyond simply data processing, numerical machine tool control, and telephone switching. But perhaps in some of these more sophisticated applications in which the computer takes over some of the human decision-making function, youngful respondents are more fearful of computers and less optimistic than adults.

Computers are a tool just like a hammer or lathe.—Again, adults are in considerably greater agreement with this statement than are younger respondents.

Computers slow down and complicate simple business operations.—Interestingly enough, most people seem to believe that computers are used reasonably well in business because 68% disagree with this statement and only 17% agree.

Computers will replace low skill jobs and create jobs needing specialized training.—Somewhat more adults agreed with this statement (71%) than did youth (62%). About 15% of both adults and youth disagreed. This implies that a substantial fear exists that computers will take a tremendous number of jobs and there will have to be a massive effort by society (retraining, welfare, or ?) to absorb the human beings put out of work by the computer monster. This leads to the next question.

Computers will replace as many jobs as they eliminate.—Again, somewhat more adults agreed (70%) than did youth (61%) and fewer adults disagreed (13%) than youth (23%). So we see that a large number of people believe the computer will replace low skill jobs, but furthermore, we see some question about the creation of new jobs by the computer to replace the ones eliminated and, as before, there is even more doubt expressed by youthful respondents.

UNDERSTANDING OF COMPUTERS

After looking at the various roles of the computer, one must ask, do people understand the computer per se by itself? And the answer is that a surprising number do. And quite a few don't. Indeed between one-quarter and one-third of the population believe the computer is beyond the understanding of the typical person. Also, many people have the wrong notion about who causes computer mistakes—machines or people.

Computers are beyond the understanding of the typical person—Are they? Well 25% of the adults and 31% of the youth think so. But 62% of the adults and 49% of the youth think they are within comprehension. Perhaps more revealing—among schools with a instructional computer program, nearly 80% of the students believe that computers are within their understanding.

Computers make mistakes at least 10% of the time—This statement must be coupled with the next one: *Programmers and operators make mistakes, but computers are, for the most part, error free.* FACT: Statement 1 is absolutely false, statement 2 is true. How did respondents do with these questions? Most answered "correctly"—about 68%, fewer youth than adults, but a fair number of people were downright wrong (13%). The rest of the people didn't know (19%). These percentages are similar to those scored on nationwide tests of scientific facts—about ⅔ of the people know the facts but the other third are wrong or just don't know. A happy situation? Not very.

It is possible to design computer systems which protect the privacy of data.—Not even the computer designer knows for sure, so what can we expect from the general public? Well, 61% of the adults think you can design a secure system and 26% think you can't; only 49% of the youth think you can and 16% think you can't. What does all this say? Probably nothing except that some people are optimists and some are pessimists, and at least on the data privacy issue, more adults are optimistic than young people.

APPENDIX—*Statistical Results of Survey of Public Attitudes Toward Computers in Society*

	ADULT (N=300)		YOUTH (N=543)	
	Strongly or Mostly Agree	Strongly or Mostly Disagree	Strongly or Mostly Agree	Strongly or Mostly Disagree
Computer Impact on the Quality of Life				
• Computers will improve education.	86.6%	5.9%	84.2%	4.5%
• Computers will improve law enforcement.	81.9	3.3	70.0	10.1
• Computers will improve health care.	78.6	5.3	54.1	11.9
• Credit rating data banks are a worthwhile use of computers.	64.2	13.4	64.0	7.6
Computer Threat to Society				
• A person today cannot escape the influence of computers.	91.6	4.0	66.6	17.7
• Computer polls and predictions influence the outcome of elections.	48.1	27.5	44.2	26.9
• Computers dehumanize society by treating everyone as a number.	37.4	50.3	39.9	30.6
• Computers isolate people by preventing normal social interactions among users.	18.7	62.5	20.9	42.5
Understanding the Role of Computers				
• Computers are best suited for doing repetitive, monotonous tasks.	80.0	10.3	57.0	21.6
• Computers are a tool just like a hammer or lathe.	72.6	14.7	61.3	23.4
• Computers slow down and complicate simple business operations.	17.6	66.4	17.4	68.8
• Computers will replace low-skill jobs and create jobs needing specialized training.	71.0	15.0	61.8	14.4
• Computers will create as many jobs as they eliminate.	62.5	16.4	40.0	29.1
Understanding of Computers				
• Computers are beyond the understanding of the typical person.	25.2	61.6	30.6	49.2
• Computers make mistakes at least 10% of the time.	9.6	76.7	10.3	60.0
• Programmers and operators make mistakes, but computers are, for the most part, error free.	67.0	19.3	72.3	13.3
• It is possible to design computer systems which protect the privacy of data.	60.2	26.4	48.6	15.9

Survey of public access to computing

by CAROL H. KASTNER and WILLIAM G. UNDERHILL
Human Resources Research Organization
Alexandria, Virginia

ABSTRACT

The Center for Inquiry and Discovery in Washington, D.C., is planning to establish a public computer center to be used primarily by school age children. A survey of existing public-access computing facilities throughout the country is being made in order to seek information and advice to assist in this planning. This paper describes the survey being undertaken, the institutions being studied and the findings to date.

SURVEY OF PUBLIC ACCESS TO COMPUTING

There is at present no public access computing facility in the Washington, D.C. area. Although universities and even some secondary schools have computers for student and faculty use, access to these systems is generally restricted. The Center for Inquiry and Discovery, a newly established institution in Washington, is planning to create a public computer center for a wide variety of uses.

The Center will organize computer-related workshops, classes, clubs and special projects for individuals, community groups and local schools. In addition, the computer will be used in the exhibit areas of the Center. There will be an exhibit devoted to computers, where the casual visitor will be able to learn something about the nature of computing and perhaps become interested enough to enroll in a class or workshop. The computer will also be used in some of the other exhibit areas to help to explain the subject matter using simulations or graphic displays where appropriate. Finally, a data bank describing community resources, available materials and upcoming events of possible interest to Center visitors will be available from terminals located throughout the exhibit areas.

In order to plan for this computer facility, a survey is currently being conducted by the Center to study similar programs at museums, science centers and other public institutions throughout the country. Also included in the survey are several research oriented labs (SOLOWORKS at the University of Pittsburgh, the LOGO project at MIT, Xerox PARC in Palo Alto) which have as a common goal the development of computer systems which can be used effectively by people having widely divergent backgrounds and interests.

Information is being collected about hardware, software, educational programs, staffing, types of users, costs and financing at these institutions. The following is a partial list of questions taken from a questionnaire being used in the survey.

Ties with other institutions

- What kinds of services are provided to local schools, colleges or community groups (e.g., field trips, classes during school hours, facilities for students to do individual projects, in-service teacher training)?
- Do students and teachers receive credit and release time for their participation?
- Does the center provide a time-sharing service for the schools or the community and, if so, what level of support is provided (e.g., development of curricular materials)?

Exhibits

- What philosophy about computing is shown to the one-time visitor?
- How are exhibits designed that are easy for visitors to use and understand, and which need minimal supervision?
- Is game playing the most effective use of computers in an exhibit area?

Educational programs

- What classes, workshops, and clubs are available for the public?
- Are educational programs provided for various age groups and levels of sophistication?
- How are educational programs, classes, workshops evaluated?

Equipment

- What kind of computer equipment (e.g., central system, terminals) is used?

- How is maintenance handled?
- What software packages are available for users?

Staffing

- What level of staffing is needed to support the computer center and its activities?
- To what extent does the center rely on students and other volunteers to perform staff functions (e.g., teaching classes, systems development, hardware maintenance) ?
- Are volunteers reimbursed in some way for their efforts (e.g., receiving school credit, revenue from sales of software) ?

Financing

- Is the computer center self-supporting or does it rely on general admission charges, grants or donations?
- Is the computer used to raise money by providing such things as mailing list and accounting services to other non-profit institutions?
- Is computer time sold to the public?
- Is tuition charged for classes and workshops?
- Are payments received for services provided to schools?

The following is a brief description of those institutions which have already been visited.

Lawrence Hall of Science on the University of California Berkeley campus is a science museum with a large staff of students. It has five computer terminals in the exhibit area running games and simulations and an extensive educational program of classes for school children and the public. Its three computers provide a low cost time-sharing service for over 35 educational institutions in Northern California.

Oregon Museum of Science and Industry (*OMSI*) in Portland has in addition to its exhibit area a community research center. The goal of this research center is to provide the community with access to otherwise unavailable scientific equipment. It has a PDP 11/45 that approximately 40 students per year use to do individual projects for which they receive high school credit. Advanced software development projects have earned revenue and equipment donations for the center.

Community Computer Center is a storefront computer center in Menlo Park, California, that offers the public access to recreational uses of computers. Their activities include a hardware club, classes for teachers, birthday parties, field trips, and game nights.

The Exploratorium in San Francisco is a large science center that hopes to use computers in the future to simulate scientific phenomena.

Pacific Science Center in Seattle is in the American exhibit building of the 1962 World's Fair. They are still planning their computer activities, but do have two terminals running games with time donated by various schools and businesses.

Boston Children's Museum has a computer, terminals, and calculators in an exhibit area for visitors and school children on field trips to use. They also use their computer for administrative purposes.

Boston Museum of Science has a large exhibit area with a computer and terminals donated by Honeywell.

Franklin Institute in Philadelphia has a terminal in the math area, running a demonstration package. Computer time is donated by businesses and schools. They also hold a summer computer workshop. They are planning to use the computer in a new Bicentennial exhibit.

Maryland Science Center is a new institution not yet open to the public. In a preview exhibit they used the computer-controlled turtle and music box developed at MIT's LOGO Project.

The LOGO Project at MIT has developed new ways for young children to learn mathematics and logical thinking using the LOGO language, computer-controlled devices and graphics.

SOLOWORKS at the University of Pittsburgh is working to develop the hardware, software, and courseware for an open mathematics laboratory. They use a wide variety of computer peripherals such as graphic displays, robots, and a pipe organ in their lab.

Xerox PARC has developed a computer system and a language called SMALLTALK which makes graphics and animation easy to use. They hope to open their own storefront computer center in the near future.

Other institutions which have been or will be contacted by telephone and visited in the future include: Ontario Science Center, Chicago Museum of Science and Industry, The Science Museum of Minnesota, Science Museum of Roanoke Valley in Virginia, Denver Children's Museum, Fernbank Science Center in Atlanta, and the Brooklyn Children's Museum.

While each site surveyed has its own unique features, certain ideas, programs and problems seem to be almost universal. Although the survey is not yet complete, a summary of the most common and most important points of information gives a picture of a "typical" public access computer center.

Coordination with local public schools is essential. The schools are the primary sources of users for a public access computer system. While the computer exhibit areas attract the attention of adult and child alike, the more in-depth behind-the-scenes activities such as workshops and computer programming classes, mainly attract school children. Many of these classes are official school functions, with the computer center being the site for repeated field trips from the schools. Payments from schools for use of the center's resources are often an important source of income.

Cooperation with local universities is important since college students are a valuable source of part-time or volunteer staff. The possibility also exists to offer in-service training for teachers in the computing center for college credit.

Most of the institutions visited have computer terminals in an exhibit area. Simple games are the main mode of initially presenting computers to visitors, however, neither staff nor user interest is sustained if the only computer activities available are games.

Cathode ray tubes (CRTs) or silent printer terminals are preferable in the exhibit areas to decrease noise. Since the majority of museum visitors tend to read little of the printed text accompanying exhibits, graphics displays are an additional advantage. Simplified or color-coded keyboards are sometimes used, particularly with young children.

The level of staffing at the centers surveyed varies from the Lawrence Hall of Science with over 30 part-time staff members, most of whom are Berkeley students, to other institutions with only one part-time person. It is important to have at least one paid staff member whose responsibility is to coordinate and provide direction for all the various activities in the center. High school and college students who develop a strong interest become a valuable, but temporary, resource as instructors, programmers, and maintenance engineers.

All the institutions surveyed have received loans or donations of either computer hardware, computer time, or terminals. One institution has traded marketable software packages to a computer manufacturer in return for equipment (e.g., a disc drive). Although donations and loans were instrumental in the development of the various centers, they also have disadvantages. Donations of computer time are often short-term commitments. Software packages developed for a borrowed computer system must either be re-implemented or discarded when the hardware base changes. Donated equipment often presents a maintenance problem since the equipment tends to be several years old. Maintenance contracts are, therefore, fairly costly and parts are more difficult to obtain.

None of the centers is totally self-supporting. Most rely on general museum admission fees, grants, endowments and donations. These revenues are often supplemented by income from school field trips, classes, workshops, and sales of computer time to individual community members and groups. Many of the museums also use their computers to do their administrative work and have expanded to provide these services to other cultural institutions at low cost.

When the survey is complete, our goal is to have developed a model of an ideal public access computer center. It is our hope to build this center in phases as we acquire the necessary funds, personnel and experience. We envision it to be a non-profit, self-supporting computer center which will be able to grow as community acceptance and demands increase.

Building your own computer

by STEPHEN B. GRAY
Amateur Computer Society
Darien, Connecticut

ABSTRACT

Microcomputer kits have changed the build-it-yourself hobby dramatically. Until the introduction of such kits, only about two dozen amateur computers of any real complexity were in operation; nearly all were built by engineers in the computer industry. Formidable problems exist when trying to use or rebuild obsolete vacuum-tube or transistor computers, such as the non-availability of schematics. Building from scratch is so complex that usually only electronics engineers achieve it. Building a machine that uses the instruction set of a commercial computer is popular. Microprocessors were introduced only a few years ago, and already two dozen microcomputer kits are available, ranging from inexpensive minimal machines using assembly language, to high-level-language systems costing several thousand dollars.

INTRODUCTION

Microcomputer kits have been available for only a little over two years, but they have changed the build-a-computer hobby quite dramatically. Up to the beginning of 1974, only about two dozen amateur computers of any real complexity were in operation, and nearly all had been built from scratch by engineers in the computer industry. Today two dozen different microcomputer kits are available, based on half-a-dozen different microprocessors, and about 7,000 have been sold, a great many to people with apparently little or no knowledge of electronics.

Despite all the activity in microkits, many computer hobbyists are still building machines of their own design, or copying a commercial machine (or its instruction set), or operating and/or rebuilding obsolete computers. Before looking at the microcomputer scene, let's first examine what can be done, and has been done, in the more individualistic areas.

USING OBSOLETE COMPUTERS

Vacuum-tube computers are occasionally available, but most of the drawbacks in using them are formidable: many are so large that a barn is required to store them, they need a great deal of air-conditioning and electrical power, and tubes can be expensive to replace.

Schematics are needed to get the computer working and maintained, but they are almost never available. Even with the older transistor computers, schematics (and especially updated schematics) are usually impossible to obtain. Now and then the prototype of a recent computer can be bought cheaply, but again, usually without schematics, so the buyer has two choices: take months or years to trace out every connection, or rewire most or all of the machine.

Several hobbyists who have bought obsolete computers with ailing or missing magnetic-drum or core memories have tried replacing them with semiconductor memory, quite a task even for an electronics engineer, which not all of them are.

Despite all the problems, many hobbyists are still operating an old computer, or trying to get it into shape, including machines such as the LGP-30, RPC-4000, LGP-21, and G-15. The CDC 160A appeals to a number of amateurs, but few if any are in the hands of hobbyists, because the machine is still relatively expensive, and it has 1700 discrete transistors, leading one California amateur to note, "I look at this number as an indication of the problem I would have in keeping it running or even getting it going." He bought an RPC-4000 "at a graveyard-type disposal sale," and later noted, "My RPC is working but I can't get an assembly program more than two-thirds loaded. This produces lots of messages telling me my programs are bad. I suspect some memory aberrations, but the memory print routine won't print either. So I have been trying to write a simpler routine of my own in machine language. That is a drag. It is amazing how many ways you can make mistakes with 32-bit instructions."

An Indiana hobbyist bought a Univac O File Computer as scrap, with arithmetic unit, program-control unit, 90-column reader/punch, sort-collate unit, tape-drive program controller, and six magnetic-tape units. The new owner says, "I had figured to use the outside winter air to get it turned on and see what I've got, and just close down in summer. As to space, not too bad:

only about 400 or 500 square feet, pretty compact. I'm presently having 220 V installed to begin to turn on some of it." The manufacturer told him they can't provide schematics for any machine this old. Each machine was somewhat different, various changes having been made to each during its life, and careful documentation had to be kept as to what was inside each. Many of the old schematics and documentation have been thrown out, and "no amount of money" could provide relevant schematics for most of these old machines, antiques at 19.

Manuals are (or were until recently) available for the Univac 1108: 20 to 30 of them, each costing $50. Some Univac Solid State computers were given away to schools; when the schools asked the manufacturer about documentation, the situation turned out to be "impossible," as there were no records available on updated blueprints. "Maintenance in those days was a tricky thing," said Univac, "and the man who did it has long since been assigned to newer equipment, so there is nobody available from us today who knows how to service the old machines."

One company was getting rid of its Univac I, and wanted to give it away. But Univac found that to take it apart carefully and reassemble it elsewhere would cost $100,000, so the machine was scrapped.

BUILDING FROM SCRATCH

As in amateur radio, many computer hobbyists would never think of buying a kit or an assembled machine; they must build one. Up until only a couple of years ago, this task was so difficult that only a couple of dozen computer hobbyists in this country had operating digital computers that could really be considered as computers, and nearly all were electronics engineers in the data-processing industry.

The main problem in building a computer from scratch, without resorting to a microprocessor, is that so many areas of specialization are involved: logic, input/output, memory, peripherals, and mechanical skills such as packaging, back-plane wiring, metalworking, plastics, and many others. To build one's own computer once required learning a great deal about each of these fields. Although some computer hobbyists are engineers who design their own circuits, most non-engineers must rely on published information, and although several dozen books and manuals contain computer schematics, they have serious limitations. A book may show schematics of various portions of a computer—arithmetic unit, memory, control circuits—but none show how to connect them together, and anyway, they are usually only partial schematics. Minicomputer manuals with full schematics can be bought, but many of the parts are identified only by the manufacturer's code number; copying such a machine would be no real hangup for a computer engineer, but it is indeed for the neophyte.

Even supposing that an amateur computer-builder did get hold of complete schematics and all the parts, the one big stumbling block that has thrown many is core memory. It is still expensive to buy when new, and when surplus it may contain broken cores, or perhaps it's surplus because it couldn't pass the manufacturer's quality control. Getting a core memory to work still separates the men from the boys, if there are still any who want to try it, now that semiconductor memory is so readily available and so cheap.

Surplus computer printed-circuit boards have been available for years, but most of them, especially the IBM SMS series, have had their "tab" ends broken off, to make sure the boards won't find their way back into commercial computers. Many of the 3800 different types of SMS boards are level-changing circuits, of little use to the amateur.

DESIGNING YOUR OWN

Computer engineers often like to design their own machines, perhaps for the challenge of making it work. One Pennsylvania engineer's pre-IC machine is 7 feet long, 1½ feet deep, and 6 feet high; it took a year to build and "will take 10 years to program," according to the builder, who used NOR gates from a process-control system declared scrap, mounted on 120 circuit boards with 35-pin connectors.

One North Carolina hobbyist built 75 percent of his computer with IBM SMS cards, the rest with home-built cards, an IBM 1620 core memory, Selectric typewriter and paper-tape reader/punch for input/output. The six-register machine has 16 instructions. "I/O devices available but not connected: 384K-word drum, two 7330 tape drives, two 100-cpm card readers. . . . A home-built line printer is ⅓ complete; 52-character chain, about 200 lpm."

A letter published in the April 1969 issue of the Amateur Computer Society Newsletter, from an ACS member now in Israel, said, in part, "In past issues of the Newsletter, some rather ingenious instruction sets have been devised which either simplify hardware, decoding or subsequent programming. It should be borne in mind, however, that the use of an instruction set which is already implemented on a commercial machine means a great reduction in problems with software, which would then be readily available. Remember that commercial manufacturers also look for instruction sets which tend to optimize both hardware and software, and many machines have instructions worth copying. If you've never written an assembler or Fortran compiler, don't just laugh it off as an easy project; it may well take you longer than to build the machine itself. Coming up with a new, unique instruction set may be a thrilling idea, but getting someone else's instruction set to function with your hardware is no small feat either."

COPYING A COMMERCIAL MACHINE

The computer amateur who wants to make things easier on himself by copying has two choices: he can obtain the schematics of his favorite computer and try to duplicate it, or he can build a machine of his own design that will use the instruction set of a commercial machine; the latter choice is the most common.

Most of those who have borrowed an instruction set already in use are copying that of the PDP-8 family, because of the variety of programs available, and a simple yet powerful set of instructions.

An Arizona hobbyist has built two computers from scrap parts. "Both are 12-bit, 2-μsec machines patterned after the PDP-8 instruction set. The first was built from second-generation discrete-component DTL NAND logic. The memory was of my own design. My second computer was built to get around the power dissipation problem (1.5W) of the first machine. It gets expensive to operate and refrigerate that kind of system in Arizona. The second machine is made out of 7400-series TTL and has an 8K x 12 main core memory. . . . All the PDP-8 software works on my system. This has saved considerable time, as you can well imagine. I have used the following DEC software: compilers (Focal—8K, Basic—Poly, Fortran—8K); assemblers (Macro 8, Pal III, Saber); maintenance programs, disk monitor systems (my 32K core memory looks like a DF32 Disk System). . . . I have devoted most of my spare time for the last four years in accumulating the parts and developing my software."

In Maryland, a computer amateur says, "Over the past four years, I have been in the process of building a computer. The actual hardware work got under way about three years ago. The machine really started working only a year ago. My machine uses the PDP-8 command set and runs at about the speed of a PDP-8/S (24-μsec cycle time). My memory is from an IBM 1620. . . . I have implemented only 4K at present, though I have designed the boards to allow easy expansion to 8K. . . . I have copies of the DEC software, which all seem to run: Focal, Editor, Pal III, etc. While I use the DEC sofeware, I have made a point of never looking at their PDP-8 hardware diagrams, etc. I'm sure I learned more this way. After all, I'm supposed to be an EE."

A Floridian built a "Mininova 721," which he designed "to be a miniature Nova 1200. . . . I wanted to have a Nova for my very own, but couldn't afford it. I thought 'someday when the price of ICs comes down I'll design and build my own minicomputer, a small-scale version of the Nova.' Well, the prices of ICs came down tenfold or more in 1971-72, and this made my dream practical. The rest was innovation, enthusiasm, and a lot of careful planning. . . . I've incorporated very carefully the best features of the Nova instruction set and programmer's console, and designed the circuits to make a true stored-program, programmable digital computer (complete wtih loads of integrated-circuit MOS-RAM memory) that would execute 16 different very carefully selected instructions. . . ."

TWO PRE-MICROPROCESSOR KITS

Back in the Sixties, the Tesla Research Foundation, with offices in Utah and Arizona, offered a variety of analog and digital computer kits, plans for digital gadgets, and home-study courses. The DI-TR5 digital computer, for instance, cost $365 in kit form, $440 assembled, used germanium transistor NAND logic and diode OR gates, and had two registers and 15 instructions. Input/output was with switches and lamps. The company didn't last long, and has vanished without a trace.

Beginning about five years ago, the National Radio Institute has been offering a course in computer electronics. The course includes building a simple desktop computer, which weighs 16 pounds. The Model 832 NRI Digital Computer contains 52 TTL ICs, 7400-type. The specifications include: 17 storage locations for 8-bit words, expandable to 32 words; over 15 basic instructions; input/output is with switches and lamps. For teaching purposes, the memory consists of slide switches.

THE MICROPROCESSOR REVOLUTION

All the previous quotes from amateurs building their own computers are from before 1974, the year in which hobbyist microcomputer kits first appeared on the market.

The avalanche started somewhat slowly, two years before, when Intel Corp. introduced two "Micro Computer Sets," the MCS-4 and MCS-8, sets of LSI chips for microprogrammable general-purpose computers. The MCS-4 was built around the 4-bit 4004 microprocessor, with 45 instructions; the MCS-8, around the 8-bit 8008, with 48 instructions. Although highly attractive to the hobbyist, both sets were expensive. The MCS-4 consisted of the 4001 programmable ROM memory, 4002 RAM data storage, 4003 I/O expansion, and the 4004 MPU. The last three were fairly cheap: $50, $10, and $100, respectively (in 1972), for 1 to 24. The catch was the 4001: the minimum order quantity was 25, at $25.50 each, plus mask charges of $600. The purchaser could have the ROMs programmed by Intel, or do it himself with the SIM-01 microcomputer (then $400), three control-program ROMs at $101 each, and an ASR33 Teletype.

The MCS-8 cost even more: the 8008 was then $200 for 1 to 24, and programming the ROMs required the SIM8-01 at $900, plus the control ROMs and Teletype.

Hobbyists soon seized upon these MPUs. A Connecticut computernik put it this way, "Enter the Intel 4004 and 8008 CPU on a chip! Both are complete

CPUs with quite a bit of power (45 instructions) and flexibility (internal address stack for subroutine nesting, etc.). The 4004 is not as desirable since it is more complicated to control and doesn't look as much like a typical computer. The 8008, however, is a beaut! . . . The only drawback I see on these devices is their slow speed (about 1 MHz), yielding about 75K instructions per second. For amateur (and many commercial) uses this should be no real problem. Whether we wait 1 sec or 3 sec for the answer does not really matter. But a cost of $5K or $1K does matter!"

MICROPROCESSOR HOBBY KITS

In early 1974, Scelbi Computer Consulting offered the Scelbi-8H modular computer kit, based on the 8008 MPU. There was a starter set of five cards—CPU, DBB (data bus buffer) and output, input, front-panel card, and RAM card (256 8-bit words)—at $440. One step up, the standard card set, with 1024 words of RAM memory, was $565. A standard computer, with cards, chassis (console switches, card sockets, input/output and power connectors), and separate power supply, was $795 in kit form, $950 assembled. The 8H deluxe had 4096 words of RAM memory, and a higher-rated power supply; $1400 in kit, $1600 assembled. The memory could be expanded to 16K words, for about $2760 more.

Peripherals for the 8H included an oscilloscope alphanumeric interface, audio cassette tape unit interface, ASCII keyboard, and bit-serial interface for Teletype.

The Scelbi 8H was manufactured from March 1974 to June 1975, and was superseded by the Scelbi-8B, manufactured from April to December 1975, and using Intel 2102 RAMs "which allow the 8B to be directly expanded up to 16,384 words of memory at a cost comparable to that of 4,096 words of memory in an 8H," which used the Intel 1101 RAM. The 8B kit, with 8008 MPU and 1K memory, was $499; empty 4K RAM card, $49; eight 2102 RAMs, $59. Scelbi provided extensive assembly-language software for their machines.

Scelbi got out of the hardware business at the end of 1975, and has since devoted itself exclusively to software, and to publishing books such as the $19.95 "Machine Language Programming for the 8008 (and Similar Microprocessors)." (The book could just as well have been called "Assembly Language Programming, etc.," but the author felt that people with little or no knowledge of computers would be more likely to understand the title the book was given.) At this writing, Scelbi is working on software such as BASIC for the 8008 and the 8080, and is considering what other MPUs to write software for.

The July 1974 issue of *Radio-Electronics* heralded the debut of the Mark-8 microcomputer kit, based on the Intel 8008 and using 7400-series TTL ICs. A minimum Mark-8, with 256 8-bit words, was about $300. The Mark-8 was available from June 1974 to December 1975, and will be superseded this Spring by the Dyna-Micro kit, an 8080 "microcomputer learning system, not a number-cruncher," with a 1702A PROM that loads on reset, and a memory of 256 words, maximum 512 words; about $200.

The January and February 1975 issues of *Popular Electronics* unveiled the Altair 8800 computer kit from MITS, based on the Intel 8080 MPU, an 8-bit-word/16-bit-address machine with 78 basic instructions. The 8800, through extensive advertising, has become the best-known and most widely sold hobby computer, with some 6,000 sold as of this writing. The basic Altair 8800, with CPU board, front panel, power supply and expander board, is $439 kit, $621 assembled. The 4k dynamic memory boards are $195 kit; the 2K static memory kit is $145; 1K static memory kit, $97; maximum memory is 65K. There are serial and parallel interface boards.

Altair hardware options include a terminal with built-in audio cassette interface, a floppy-disk system, and a line printer. Software includes 4K and 8K BASIC, 12K Extended BASIC, assembler, text editor, system monitor, debug and DOS.

Sphere, a Utah company, offers the 4K Sphere 1 computer kit, using the Motorola 6800 MPU, with 512-character TV terminal, keyboard and power supply, for $860 ($1400 assembled). Memory is expandable to 65K, at about $240 for a 4K board, $400 for 8K, $750 for a 16K memory-board kit. A mini-assembler, editor, debugger, and utility commands are built into 1K of PROM. Available software includes Extended BASIC (with string and matrix manipulation), machine-language subroutine calls, trig functions, and disk-file input/output, plus FDOS (flexible disk operating system). The Sphere 2 kit adds serial communications and audio-cassette capability, at $999. The Sphere 3 kit adds 16K more of memory, for $1765. The Sphere 4 kit includes a 65-lpm printer, two IBM-compatible floppy disks, and DOS, at $6100. Other Sphere products include a light pen, and both full-color and black-and-white video graphics systems.

The MITS Altair 680 kit, with the Motorola 6800 MPU, is less than a third the size of the Altair 8800. Software includes a monitor on PROM, assembler, debug and editor. The 680 has three interrupt levels (the 8800 has eight). A $345 kit includes 1K bytes of RAM; the price for the 12K RAM board kits has not been set as of this writing.

The SWTP 6800 kit, from Southwest Technical Products Corp., using the Motorola 6800 MPU, comes with serial interface, 128 words of static scratchpad RAM, and 2,048 words of main memory, at $450. Additional 4k memory boards are $125 each; interface cards are $35. The SWTP 6800 contains a "Mikbug" ROM with a program that allows data to be entered the moment power is turned on.

In addition to the Altair 8800 and 680, Sphere and

SWTP 6800 kits, over a dozen others are available, including the Mike 2 and Mike 3 (Martin Research), Mark 80 (E&L Instruments), MOD 8, MOD 80 and RM6800 (MiniMicroMart), 008A (RGS Electronics), "George" (Godbout Electronics), Micro 440 (Comp-Sultants), SRI-1000 (Systems Research Inc.), Imsai 8080 (IMS Associates), Jupiter II (Wave Mate), and Micro-68 (Electronic Products Assoc.).

Some kits are based on the Intel 8008 (Mike 2, MOD 8, 008A), or 8080 (Altair 8800, Mike 3, Mark 80, MOD 80, Imsai 8800), others on the Motorola 6800 (Altair 680, Sphere, SWTP 6800, RM6800), National Semiconductor PACE ("George," SRI-1000), or Intel 4040 (Micro 440).

Many microprocessors are available as part of a PC-board kit marketed for engineering evaluation, usually with a minimum of memory, and without power supply, chassis or case. Some of these are being bought by hobbyists, including the JOLT from Pehaco Corp., built around a MOS Technology 6502, at $249; a Mostek F8 Evaluation Kit, using the Mostek 3850; three evaluation kits from Cramer Electronics, based on the Intel 8080A, Texas Instruments 8080, Motorola 6800, $495 each. Cramer has planned later evaluation kits based on the AMD 9800, Mostek F8, RCA COSMAC, and bipolar MPUs such as the Intel 3001, AMD 2901, TI C400, Motorola 10800.

KIT PROBLEMS

Building a computer kit is not all that simple for a beginner. One kit manufacturer says, "For every person who can wire a computer kit, there are ten who can't." But those ten do try. Another manufacturer says, "Half the people buying kits are not qualified to build them. From the ones that are sent back, it's obvious that most people don't know how to solder, don't even know how to put components on boards. Even resistors get all jumbled up. Our literature says the builder should have a couple of years experience in electronics, but people just don't believe that."

Because of these problems, Sphere is now offering the Micro-Sphere 200, in assembled form only. Built around the 6800 MPU, the basic 200 comes with 4K RAM, cassette loader, cassette operating system, 12-line by 21-character alphanumeric character generator, and "Monte Carlo games package," at $860. A second 4K of memory is $180; an Extended Business BASIC ROM, $400; floating-point and trig ROM, $130.

The latest hobbyist computer offered by Systems Research is the SRI-F8, sold as a wired unit only, not a kit, because "too many people building computers have blown CPU chips. It's too dangerous." Quite a few beginners (and some professionals too) have destroyed sensitive ICs with static electricity, and have also put ICs in the wrong sockets or wrong-way-around, and burned them out. The SRI-F8, with the Mostek F8, 1K words, Teletype interface and debug program in ROM, is $325. Options are power supply, cassette interface, keyboard interface, video interface, keyboard, and enclosure.

Not all marketers of microkits are altruistic to the point of really trying to provide a top-quality product with adequate assembly and operating instructions. A few kits are little more than a box of parts, delivery of some kits is on an uncertain schedule, and some manuals that are supposed to be for beginners, would be understood only by a computer-design engineer. Surely many beginners are buying kits with no knowledge of what is involved in writing programs in assembly language, which many will find to be tedious, uninspiring and error-prone. And there are only a couple of books or manuals that teach the tyro how to use microprocessor assembly language; nearly all the manufacturers' publications are meant for the engineer or the professional programmer. Of course some computer enthusiasts thrive on assembly language, but many beginners are likely to soon find that what they really want is a high-level language such as BASIC. So far, BASIC is available on only a few hobby computers, including the Altair 8080, Sphere, and Jupiter II.

AMATEUR COMPUTER CLUBS AND PUBLICATIONS

A variety of amateur computer clubs and newsletters exists. The Amateur Computer Society, the oldest, has been publishing a newsletter since 1966 (all the quotations regarding hobbyist computers are taken from the ACS Newsletter). The other newsletters, and the clubs, came into being after the introduction of microprocessor kits, and include the Micro-8 Newsletter (Lompoc, Calif.), Homebrew Computer Club Newsletter (Menlo Park, Calif.), Interface (Studio City, Calif.), and The Computer Hobbyist (Cary, N.C.). The Amateur Computer Club Newsletter first appeared in England in March 1973; the newsletter of the Association Francaise des Amateurs Constructeurs d'Ordinateurs was first published in the Spring of 1974.

THE LAST QUESTION

One big question remains for the amateur: after you've gotten tired of playing games on your kit computer, what are you going to do with it? The discussion of this question opens up a whole new area, so large that another paper should be devoted to it.

THREAD (Three-dimensional Reconstruction And Display) with biomedical applications in neuron ultrastructure and computerized tomography

by JOHN C. MAZZIOTTA and H. K. HUANG
Georgetown University Medical Center
Washington, D.C.

ABSTRACT

THREAD is a computer system capable of displaying three-dimensional images from serial two-dimensional sections. The serial two-dimensional input images can either be in the format of 35 mm films or in digital cross sectional pictures of live patients generated by the ACTA (Automatic Computerized Transverse Axial) Scanner. The hardware components of the system consists of FIDAC (Film Input to Digital Automatic Computer), MACDAC (MAn Communication and Display to Automatic Computer), and the Display Subsystem of the ACTA-Scanner. The software includes such features as segmentation, rotation, hidden line removal, shading plane definition, shading and display. Two examples, one from serial electron micrographs of a neuron and the other from serial ACTA, live patient, scans are given to demonstrate the capability of the system.

INTRODUCTION & BACKGROUND

The study of tissues in the biomedical sciences is most often in the form of two-dimensional cross sections. This is especially true of microscopic data obtained from both the electron and light microscope. The recent advent of computerized tomography, whereby computer generated cross-sectional tomograms are produced anywhere in the whole living human body, provides a new additional source of two-dimensional serial images.

Three-dimensional reconstruction of this data has for the most part been avoided because of the difficult and time-consuming techniques involved. However, the added insights and the additional qualitative and quantitative data that can be obtained from three-dimensional studies, prompted investigators to undertake three-dimensional reconstructions using graphic art techniques. These reconstructions took the form of wax and plexiglass models as well as artists' drawings.[1-11] Although these methods produced some elegant results, they were quite costly in terms of artists' fees, time-consuming and lacked simple means to obtain quantitative data such as volume, shape and surface area.

To obviate these disadvantages, computer graphic systems were employed to provide semi-automatic three-dimensional reconstructions of microscopic data.[12-17] Most of these systems required some type of photographic manipulations of the input data in order to transform the information into digital format. These procedures included tracing photographic prints with light pens or cursors,[13,14] the use of a camera lucida followed by tracing on a data tablet [15] or manual tracking of cell outlines using the computer as a notebook for coordinate points.[16,17] Output from these earlier systems was in three forms, purely quantitative data such as perimeters and surface area,[15] schematic diagrams of reconstructed images[12,16,17] or line drawings depicting the contours of the section outlines which in most cases served as a guide for artists who then drew shaded three-dimensional images of the structure.[12,13,16,17] Most of these earlier systems provided the important advantage of image rotation so that any surface of the reconstructed object could be visualized.

In the field of computerized tomography, cross-sectional tomograms are taken allowing for the visualization of the internal anatomy of the living patient in the horizontal plane. Consecutive scans obtained at fixed intervals along the patient's longitudinal axis provide serial cross sections of the patient. Methods exist for the alignment of such data and the display of other (sagittal, coronal, or arbitrary) planes obtained from the original cross-sectional scans.[18-20] Three-dimensional imaging of selected structures, however, allows a specific object (bone, organ, tumor, etc.) to be displayed as an isolated image which can be rotated to allow visualization of all surfaces. This display is analogous to the situation encountered with microscopic serial cross-section reconstruction and is the logical next step in computerized tomography data display.

The three-dimensional reconstruction and display of nonbiomedical data has been an active field in computer graphics. The work of Roberts, Warnock, Sutherland and others demonstrates the excellent results that have been obtained in the three-dimensional reconstruction of geometric and architectural structures.[21-28] However, the reconstruction and display of three-dimensional biomedical data of the type described above, differs from geometric data in two specific ways: (1) algorithms to reconstruct biomedical serial section data can take advantage of the initial condition that all two-dimensional sections are aligned with respect to their X and Y axes and that these sections follow each other sequentially along the Z axis prior to rotation, and (2) biomedical data reconstruction in three-dimensions is complicated by the lack of any geometric assumptions about individual section outlines, since they may assume any random shape and are often quite complex.

The THREAD system allows for the three-dimensional reconstruction of biomedical data in the form of serial cross sections, using existing hardware devices and methods analogous to earlier three-dimensional geometric or architectural reconstructions. The system locates objects of interest in each section automatically, thereby eliminating time-consuming manual outline tracing. THREAD can rotate objects to any viewing angle and the system generates both contour-grams (line drawings), as well as fully shaded, three-dimensional images in both color and black-and-white in a fast and highly automated fashion.

The major hardware subsystems of THREAD include: FIDAC[29,30] (**F**ilm **I**nput to **D**igital **A**utomatic **C**omputer) a flying spot scanner for 35 mm film, MACDAC[30,31] (**MA**n **C**ommunication and **D**isplay to **A**utomatic **C**omputer) an interactive computer graphics terminal, and the video graphic display subsystem of the ACTA-Scanner.[32] The software subsystems include: programs to digitize 35 mm film negatives, find boundaries of objects of interest in each section, align section outlines, erase hidden lines and programs to rotate and shade the completed three-dimensional object.

The THREAD system can accept input data in two forms, either 35 mm film negatives of serial sections photographed from any source or serial computerized tomograms generated by the ACTA-Scanner which are stored on magnetic tape. Figure 1 depicts the complete THREAD system.

In this paper we present a detailed discussion of the use of the THREAD system using two examples: reconstruction of images from electron micrographs recorded on film and macroscopic structures recorded *in vivo* from serial computeriezd tomograms obtained with the ACTA-Scanner.

HARDWARE DESCRIPTION OF THE THREAD SYSTEM

The hardware of the THREAD system was already in existence and consists of three major components, all of which convert digital information into graphic displays in an interactive manner. Input data can be digitized from 35 mm film negatives, as is the case with photomicrographs or if the input data is already in digital form as is the case with ACTA-Scanner output tapes, this information can be directly read by the computer from a magnetic tape unit. Data is processed by an IBM 360/44 computer equipped with disc storage for intermediate results.

Film input

Serial sections of structures can be recorded on 35 mm film and converted to digital form by the FIDAC device (Figure 2). Film is positioned in this device and a flying spot scan is generated whereby the image on the film is digitized into 16 grey-levels, the value of which is proportional to the relative translucency of the film at each particular spot. The scanner can sample some 450 rows of spots with 672 spots per row, converting a standard 35 mm film frame into a grid of 450 x 672 numerical values. This data is then transmitted on-line to the computer, the whole process requiring ⅓ of a second.

The FIDAC device can also send the digital signals of its scan to the MACDAC display device (Figure 3) without storing them in the computer memory so the

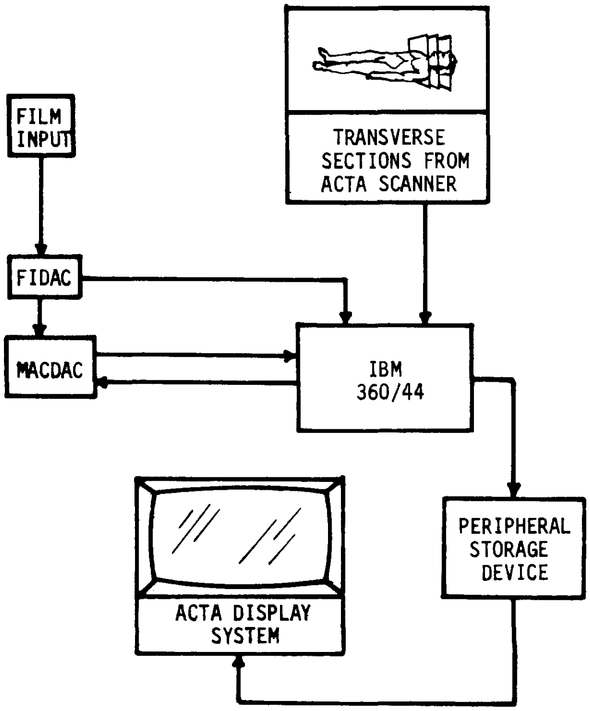

FILM INPUT
TRANSVERSE SECTIONS FROM ACTA SCANNER
FIDAC
MACDAC
IBM 360/44
PERIPHERAL STORAGE DEVICE
ACTA DISPLAY SYSTEM

Figure 1—Block diagram of the complete THREAD system

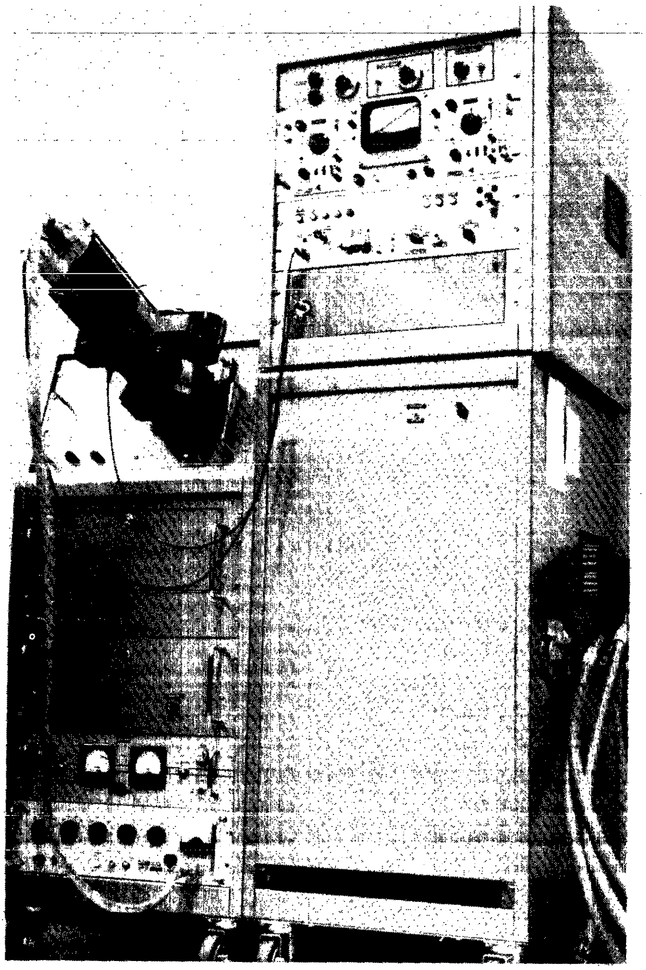


Figure 2—FIDAC: Film input to digital automatic computer

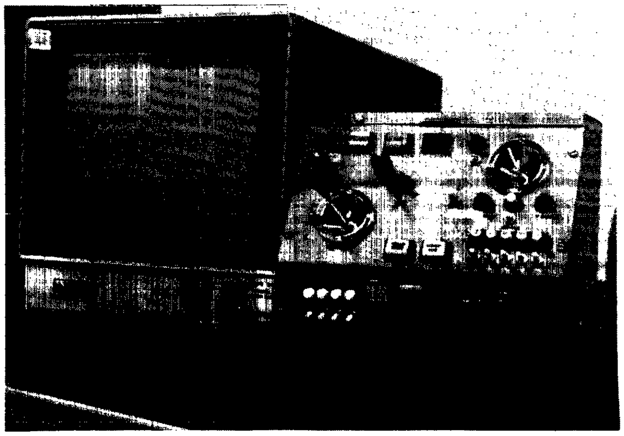


Figure 3—MACDAC: Man communication and display to automatic computer

operator can see the image and make adjustments prior to computer data acquisition.

MACDAC

The MACDAC device is an interactive computer graphic system which provides for CRT display of digital information transmitted to it by the computer or FIDAC. In addition to display, the device has the capability of transmitting coordinates from its circuitry to the computer allowing the operator to interact with and modify the image on the CRT.

The operator may point out objects using MACDAC by positioning a light spot on the screen by means of a joystick controller and pressing a "Read" button on the device which sends the point's coordinates into the computer.

Video display

Output in the form of two 160 x 160 matrix pictures is generated by the THREAD system and stored on magnetic tape for viewing on the display processor of the ACTA-Scanner, Figure 4. The output portion of the ACTA-Scanner is usually employed in the display of computerized tomograms but also provides a flexible graphics display system for information which can be stored on magnetic tape as 160 x 160 or 320 x 320 matrices.

The components of the ACTA display system include: a magnetic tape unit (Figure 4-2), a preprogrammed video display processor called the RAMTEK (Figure 4-3,4), a 19-inch color TV monitor (Figure 4–5), two 12-inch black-and-white TV monitors (Figure 4-6), a mini-computer (Figure 4-7) and a teletypewriter console (Figure 4-1).

Two 160 x 160 matrix pictures (or a 320 x 320 matrix picture) are simultaneously displayed side-by-side on all three monitors. The display system has the

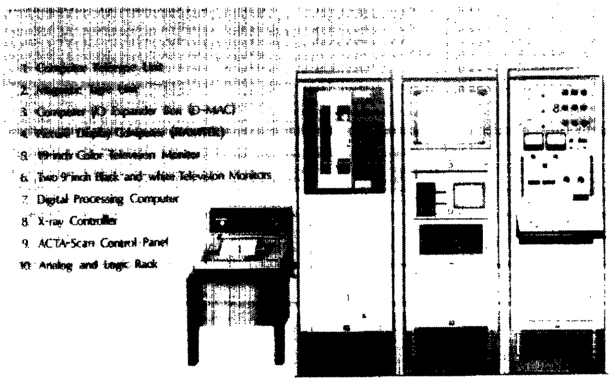

1. Computer Teletype Unit
2. Magnetic Tape Unit
3. Computer I/O Expander Box (D-MAC)
4. Picture Display Computer (RAMTEK)
5. 19-inch Color Television Monitor
6. Two 9-inch Black and white Television Monitors
7. Digital Processing Computer
8. X-ray Controller
9. ACTA-Scan Control Panel
10. Analog and Logic Rack

Figure 4—ACTA scanner video display components

capacity of distinguishing 2048 discrete intensities which are viewed as sixteen grey-levels.[33] For our purposes, the 2048 possible intensities are equally divided into the sixteen grey-levels, each with a range or width of 128 intensity values. These grey-levels may be assigned a continuous gradient of black to white or color codes may be selected for some or all of the individual grey-levels.

In the final display, a heading appears along with the two reconstructed images. The heading contains information describing the object and its orientation (Figure 13). A grey-level reference is also provided where the sixteen grey-levels are displayed as individual blocks along with the maximum and minimum intensity values represented by that particular grey-level.

The ACTA display system provides for automatic color coding, image enlargement and various photographic formats which the operator can request by means of selecting the appropriate code word on the teletypewriter.[34]

SOFTWARE DESCRIPTION OF THE THREAD SYSTEM

All driver programs are written in either FORTRAN IV or Assembly and interact with the operator in everyday language via typewriter console.

Data gathering phase

Film input data is received from the FIDAC device one frame at a time and the portion of the frame which contains the objects to be reconstructed is stored in sixteen grey-levels in the computer memory. Digital input from magnetic tape is handled in a similar fashion. Any single grey-level or composite of grey-levels may be displayed on the MACDAC monitor.

The operator is asked for information pertaining to each section including: its thickness and magnification. The operator is also instructed to point out, using the MACDAC light spot, the left-most boundary of an object of interest and three alignment points to be used later for orienting all sections in the series relative to one another. Automatic boundary location commences once the above information is obtained by the program.

Standard boundary detection methods are used: boundaries or section-object outlines which differ from their surroundings by one or more grey-level can be automatically located by the program.[35] This process is initiated in the memory when the operator points out an object of interest and types the object's grey-level into the console. The coordinates of the light spot serve as a pointer or bug which moves to the right until contacting a grey-level which equals or exceeds the threshold set for this object. The program then moves the bug clockwise around the object boundary, recording the coordinates in a 4 x 1000 element array where the first two columns represent the boundary point's X and Y coordinates, the third column represents the Z coordinate (proportional to section thickness) and the fourth column is used for later coding of selected points. The boundary processing terminates when the starting point is again located by the bug. All points are plotted on the MACDAC monitor and the operator may choose to manually edit the result if the object outline was indistinct. Up to nine objects may be located and stored for each section.

The above procedure is repeated for each section in the series and once completed, all sections are aligned using the alignment markers previously obtained for each section. Alignment is obtained by a best-fit approximation. When all objects in the series are aligned, the lists of their boundary points may be stored on magnetic tape and a new series of sections can be processed.

Reconstruction phase

The reconstruction phase of the software package is divided into six distinct steps. These six steps are preceded by the acquisition of the object boundary point lists from magnetic tape and are followed by display of the completed image.

1. *Segmentation:* Each object outline in the series is subdivided into segments of equal length, the end points of which will later be connected between adjacent sections to facilitate shading. These segment end points are assigned consecutive code numbers termed segment marker numbers. As will be seen later, the higher the number of segments chosen, the greater will be the detail of the final image.

Segmenting begins by determining the section-object outline which contains the greatest number of boundary points. This number represents the greatest possible number of segment markers available for this object. The operator is then asked what percentage of this maximum should be used in the reconstruction. For example, if the maximum segment number equals 100 and the operator chooses to use 50% of these segments, a section through this object which contained 50 boundary points would have the fourth column of its 4 x 1000 boundary point array filled with consecutive numbers from 1 to 50. If the section outline had greater than 50 boundary points, zeroes would be inserted between segment marker numbers at regular intervals (e.g. 1,2,0,3,4,0, . . ., 50) and if the object had less than 50 boundary points, segment marker numbers would be evenly deleted (e.g. 1,6,11,16, . . ., 50).

2. *Rotation:* Section object outlines are rotated by selecting two rotational angles, one about the vertical Y axis and one about the horizontal X axis. These angles are substituted in a rotational transformation

matrix and the X, Y, and Z coordinates of all object boundary points are transformed by matrix multiplication.[24] All rotated points must be checked to insure that they are positive values and if negative values are encountered, the entire list of boundary points for the object must be shifted into the positive domain by addition of a constant to the appropriate coordinate set.

3. *Hidden Line Removal:* Hidden lines are defined as portions of section-object outlines which are overlapped by the addition of a new section on top of it. Other authors have used a variety of techniques to remove hidden lines during reconstruction,[22,27,28] but solutions were directed at objects or groups of objects composed of plane polyhedra. These algorithms perform depth sorts along the Z axis at critical points and reject lines that would not be visible in the final image. However, input data for three-dimensional reconstruction from serial sections is already depth sorted, each section lying sequentially in front of the previous section. One merely examines the rotational angles to determine the forward-most section and the sequence will be known for all remaining sections. We define the forward-most section as the section last digitized unless the object is rotated greater than 90 degrees about either the horizontal or vertical axis in which case the first section to be digitized is defined as forward-most.

The procedure for hidden line removal amounts to stacking each section upon the previous section starting with the back-most and erasing any lines which fall within the area inside the forward-lying section. The process begins by initializing a region of memory to zero. This region of memory is the same size as the completed picture (160 x 160) and is called the contour-gram matrix. Processing begins by "drawing" the back-most section on this matrix, this is accomplished by locating, on the contour-gram matrix, the X and Y coordinate of each boundary point for this section and inserting the value of that point's Z coordinate in the matrix. The Z coordinate is used later in processing and serves as a code for that particular point.

Following this step, the same procedure is carried out for the next forward-lying section. This is followed by erasure of overlapped boundary points lying inside the newly added section. Erasure is accomplished by a line-by-line examination of the matrix points lying within the confines of the forward-lying section. Each point lying within these confines, regardless of its value, is replaced by a negative number, the magnitude of which is equal to the section's number in the serial set (e.g., the area inside the outline of section number 5 is coded with -5's). The use of a special negative number serves two purposes: (1) it defines areas of the matrix as within the confines of a specific section outline (a code used later in processing), and (2) it allows for simplified display of the complete contour-gram by lighting only those points on the MACDAC monitor which have positive values (i.e. the unerased Z coordinates). All sections are processed in sequence from back to front in this way.

4. *Plane Definition:* Planes are defined as rectangular areas lying between visible portions of adjacent sections and are formed by connecting corresponding segment markers. Starting with the forward-most section, a search is made of the boundary point list for the lowest segment-marker number. Once found, a similar search is made for the same number in the boundary point list of the section immediately behind the first. If either marker is missing, the program moves to the next marker number and the search is repeated.

If both segment markers are found in the arrays, the contour-gram matrix is examined to determine if these points (represented by their Z coordinates) appear in the picture or if they have been erased during hidden line removal. Three possible outcomes exist: (1) if both points are located, they are connected by a line and the program moves to the next marker number; (2) if neither point is located, the line is ignored and the program moves on; (3) if one of the two points is missing, it is dealt with as a special case.

In the special case of a missing point in the forward-lying section (Figure 5a), a line is drawn connecting the located back segment point and the expected position of the missing forward marker point. Points along this line are tested from back to front and the line is terminated when it encounters a positive number. The case is similar for a missing back segment marker except that the points along the connecting line are tested from front to back (Figure 5b).

Finally, a case exists where both segment marker

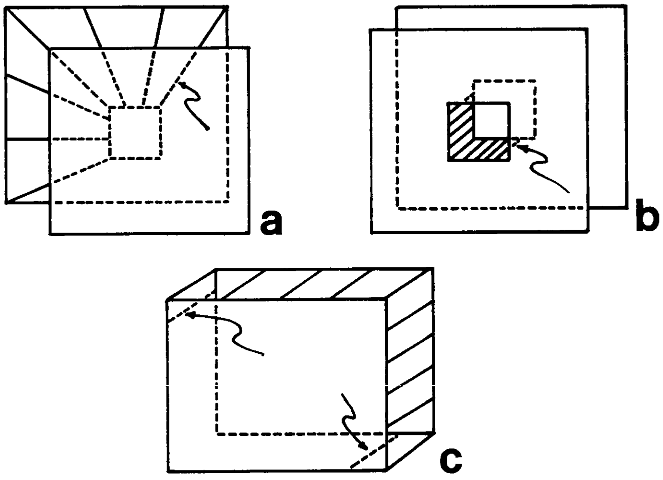


Figure 5—Special cases in "plane" determination described in text, (a) arrow shows partial line drawn in case of missing segment marker point in the forward-lying section, occurring with two large sections straddling a smaller section, (b) arrow shows partial line drawn in case of missing segment marker in back-lying section, (c) arrows show potential erroneous lines with both segment marker points visible in contour-gram (dotted lines indicate hidden lines not seen in final image)

coordinates can be located in the contour-gram but the connection of these points would be erroneous (Figure 5c). To eliminate this possibility all connecting lines are always tested to insure that their points never enter an area coded with the negative value of the forward-most section. When this occurs, the line is discarded.

After all segment markers in each section pair have been processed, adjacent connecting lines are grouped in pairs resulting in planes lying between the two sections. The program examines all sections in the series, defining planes between them and generates a list of the vertices of all planes.

5. *Shading Values:* Shading values or the light intensity assigned to each plane are derived using the methods described by Bouknight.[25] The light source and the viewer are defined as being in the center of the viewing screen and at an arbitrary distance from it. Since the graphic display system can distinguish 2048 intensities, this defines the range of shading values available. Each plane is assigned a shading value proportional to the angle it makes relative to a normal to the viewing screen. Those planes lying perpendicular to the normal are brightest and those parallel to it are darkest.

6. *Shading:* Using its respective vertices, each plane including the surface of the forward-most section is essentially "colored-in" with its assigned shading value on the 160 x 160 contour-gram matrix. The completed image and the image of the contour-gram generated for this object at the selected angles are then stored on magnetic tape for display.

RESULTS

Microscopic structures

Three-dimensional reconstruction on an ultrastructural level is demonstrated for a nucleolus obtained from cross sections through a neuron of the cat brain (Figure 7). This cell was serially sectional and photographed in the electron microscope at a uniform magnification.[36] Six equally spaced sections through the cell's nucleolus were used as input data for the reconstruction process. The physical time required to position the film in the FIDAC device, digitize the picture and locate the nucleolar outline was approximately two minutes.

As was previously described, varying the number of segment markers alters the degree of detail in the final image. This can be seen in Figure 8 where the contour-gram of two sections through the nucleolus is seen with varying numbers of segment markers connected. Image 8a was produced using only 10 percent of the available segment markers. Image 8b uses 50 percent, while Image 8c uses 90 percent of the markers. The number of planes produced in the image as well as the processing time required to generate these planes increases as the number of segment markers increase. The difference in processing time is significant and is demonstrated by the fact that Image 8c required approximately two minutes of processing time while Image 8a required only 15 seconds.

Contour-grams of this nucleolus at four different rotational angles can be seen in Figure 9. Contour-grams such as these can be displayed along with the shaded three-dimensional image at the end of processing or can be visualized during processing on the MACDAC monitor.

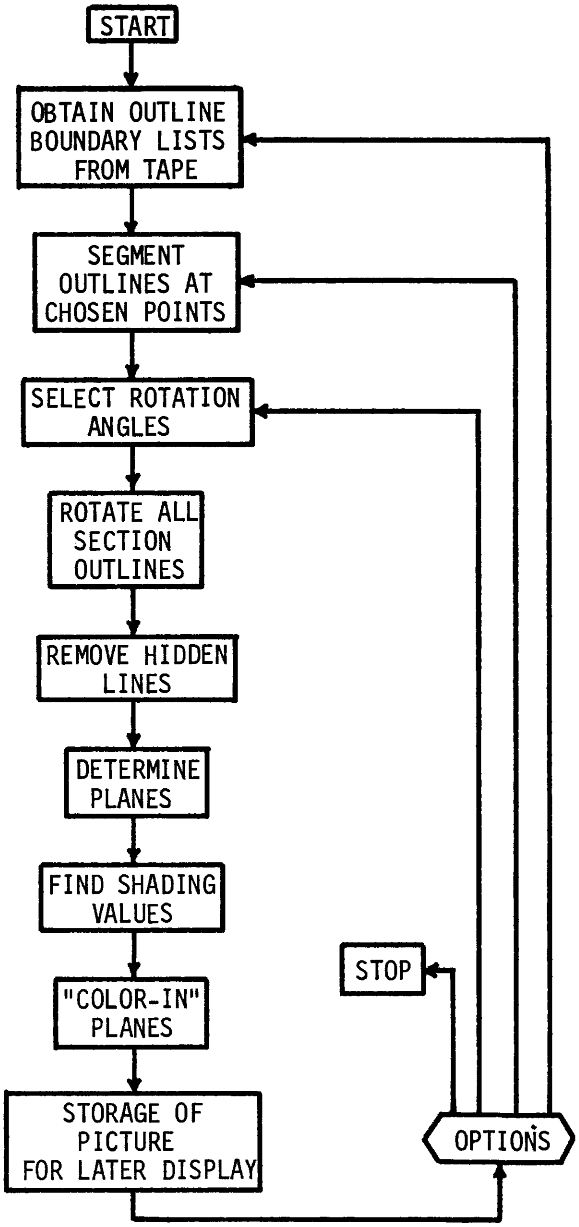

START
OBTAIN OUTLINE BOUNDARY LISTS FROM TAPE
SEGMENT OUTLINES AT CHOSEN POINTS
SELECT ROTATION ANGLES
ROTATE ALL SECTION OUTLINES
REMOVE HIDDEN LINES
DETERMINE PLANES
FIND SHADING VALUES
"COLOR-IN" PLANES
STORAGE OF PICTURE FOR LATER DISPLAY
STOP
OPTIONS

Figure 6—Block diagram of THREAD software (Reconstruction phase)

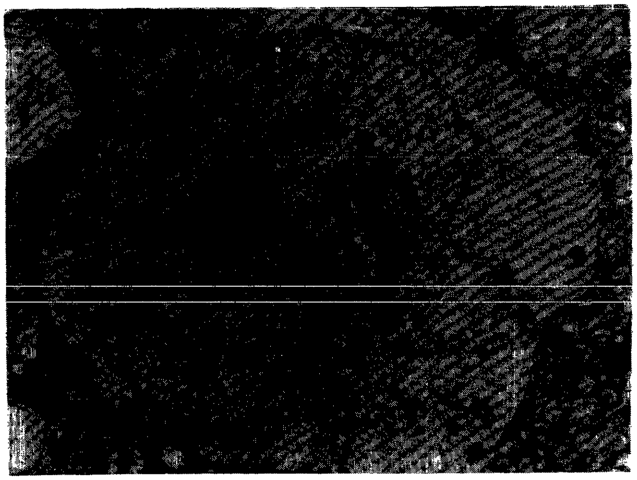


Figure 7—Electron micrograph of neuron from the cat brainstem. The dark round area in the center is the nucleolus

Figure 10 shows the final shaded image of the neuron nucleolus at two rotational angles varied with respect to the vertical axis. The structure is tapered at one pole and is slightly constricted at the center. Processing time for this image was slightly under five minutes. Image quality was consistent throughout a wide range of rotational angles.

Computerized tomography

A patient with a biopsy-proven brain tumor was referred for computerized tomography of the head. A serial set of ten ACTA scans was obtained at 1 cm intervals beginning at the level of the external auditory canal. Contrast material was used since it enhances tumor visualization in the brain. The tumor was present in five of the ten scans (Figure 11).

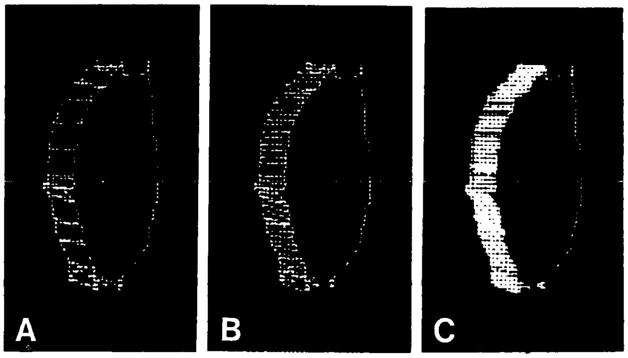


Figure 8—Contour-gram of sections through a neuron nucleolus showing the effects of varying the percentages of segment marker points used in the reconstruction, (A) 10 percent, (B) 50 percent, (C) 90 percent

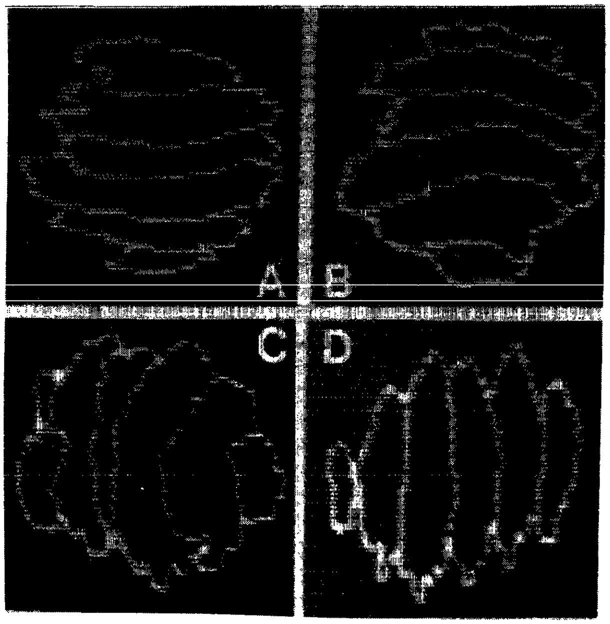


Figure 9—Contour-grams of the nucleolus seen in Figure 7 at four rotational angles

The five sections showing the tumor were recorded on magnetic tape and served as the input data to the THREAD system. Processing resulted in the shaded three-dimensional views of the isolated tumor mass seen in Figure 12. Since no tumor was seen above or below the 5 input sections, the resultant image is truncated at both poles. Thinner input sections would provide a more detailed view of these regions. Each individual picture point on the viewing monitor represents 1.5 square millimeters for surfaces parallel to the viewing screen.

Figure 13 shows the standard display format of the THREAD system including: date, object orientation and composition in coded form and the grey-level block display with reference numbers described earlier. Usually, a contour-gram and a shaded image or two shaded images at different rotational angles are viewed side-by-side on the screen. Figure 13 shows the automatic enlargement feature of the system with which the operator can type a code word into the console and have either the left or right image enlarged on the opposite side of the screen. Color formats are also available as is the capability for Polaroid photography from the monitors.

SUMMARY AND FUTURE BIOMEDICAL PROSPECTS

The THREAD system represents an effective and highly automated method for three-dimensional reconstruction and display of biomedical data from many

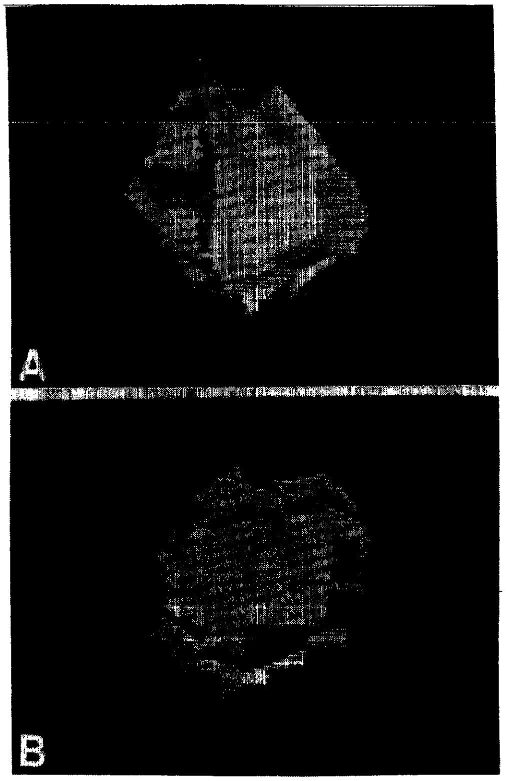


Figure 10—Shaded three-dimensional image of a neuron nucleolus generated at two rotational angles by the THREAD system: (A) right lateral view, (B) left lateral view. The program assumes viewer and light source are directly in front of monitor and assigns the brightest shades to planes lying parallel to the viewing screen

sources. It has a number of advantages over previous three-dimensional reconstruction methods, particularly the ability to automatically locate object-boundary outlines and to produce shaded three-dimensional images of the reconstructed data.

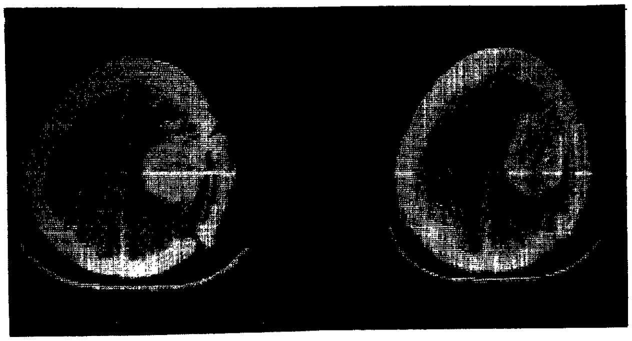


Figure 11—Two adjacent computerized tomograms produced by the ACTA scanner from a patient with uptake of contrast material by a brain tumor in one cerebral hemisphere. Skull defects are the result of previous surgery

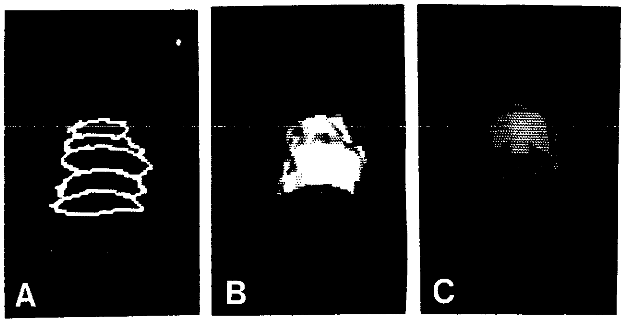
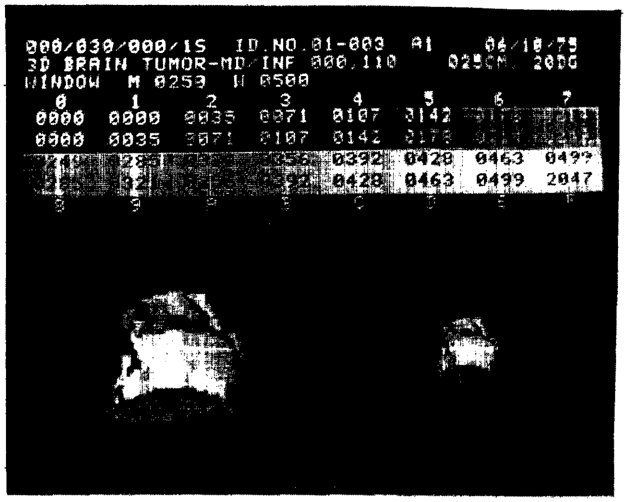


Figure 12—Three-dimensional reconstruction of brain tumor by the THREAD system, (A) Contour-gram of inferior and medial surfaces of tumor mass, (B) Shaded image of A, (C) Greater rotation of same tumor about the horizontal axis, showing more of the inferior surface

The shaded image might be further enhanced by a number of techniques to reenforce the three-dimensional illusion created by the shaded image. Smoothing of the shading planes by performing successive linear interpolations in both the X and Y direction will result in a more continuous shading gradient.[37] Stereo pairs can be generated, side-by-side, on the viewing screen with each image symmetrically rotated with respect to its vertical axis and by adjusting the relative X-axis spacing of the two images to accommodate for differing interocular distances in the viewer. In addition, movies could be made of the object at consecutively varying rotational angles thereby allowing for rapid viewing of all object surfaces.

Finally, quantitative data such as volume, shape, and



```
000/030/000/15  ID.NO.01-003  A1     06/10/75
3D BRAIN TUMOR-MD/INF 000.110     025CM. 200G
WINDOW  M 0253  W 0500
  0       1       2       3       4       5       6           7
0000    0000    0035    0071    0107    0142    [illegible] [illegible]
0000    0035    0071    0107    0142    0178    [illegible] [illegible]
[illegible] [illegible] [illegible] [illegible] 0392  0428  0463  0497
[illegible] [illegible] [illegible] [illegible] 0428  0463  0499  2047
  8       9
```

Figure 13—THREAD display format with headings, display level shading block and reconstructed brain tumor. The image on the right has been automatically enlarged four times and displayed on the left side of the viewing screen

surface area may be readily calculated for individual objects.[38] Comparison of this type of data are important in the study of changing cell morphology during experimental manipulations. Similarly, quantitative data obtained from reconstructed computerized tomograms can be used to study normal anatomy *in vivo* and to examine the effects of various treatment modalities on the size and morphologic characteristics of tumors and other pathological entities.

ACKNOWLEDGMENTS

The authors wish to thank the staff of the National Biomedical Research Foundation for their assistance and the generous use of their facilities. In particular we appreciate the initial interest and continued support of Dr. Robert S. Ledley in this project and the programming assistance of B. Shiu. We thank Dr. Betty Hamilton and the Department of Anatomy, Georgetown Medical School for their encouragement and guidance.

REFERENCES

1. Bang, B. G. and F. B. Bang, "Graphic Reconstruction of the Third Dimension from Serial Electron Micrographs," *J Ultrastructure Res, 1:* pp. 138-149, 1957.
2. Sjöstrand, F. S., "Ultrastructure of Retinal Rod Synapses of the Guinea Pig as Recorded by Three-Dimensional Reconstructions from Serial Sections," *J Ultrastructure Res, 2:* pp. 122-170, 1958.
3. Mitchell, H. C. and J. C. Thaemert, "Three Dimensions in Fine Structure," *Science, 148:* pp. 1480-1482, 1965.
4. Karlsson, U., "Three-Dimensional Studies of Neurons in the Lateral Geniculate Nucleus of the Rat, Part 1," *J Ultrastructure Res, 16:* pp. 429-481, 1966.
5. Karlsson, U., "Three-Dimensional Studies of Neurons in the Lateral Geniculate Nucleus of the Rat, Part 2," *J Ultrastructure Res, 16:* pp. 482-504, 1966.
6. Keddie, F. M. and L. Barajas, "Three-Dimensional Reconstruction of *Pityrosporum* Yeast Cells Based on Serial Section Electron Microscopy," *J Ultrastructure Res, 29:* pp. 260-275, 1969.
7. Poritsky, R., "Two and Three-Dimensional Ultrastructure of Boutons and Glial Cells on the Motoneuronal Surface in the Cat Spinal Cord," *J Comp Neurol, 135:* pp. 423-452, 1969.
8. Barajas, L., "The Ultrastructure of the Juxtaglomerular Apparatus as Disclosed by Three-Dimensional Reconstruction from Serial Sections," *J Ultrastructure Res, 33:* pp. 116-147, 1970.
9. Wiley, T. J. and R. L. Schultz, "Intranuclear Inclusions in Neurons of the Cat Primary Olfactory System," *Brain Res, 29:* pp. 31-45, 1971.
10. Dunn, R. F., "Graphic Three-Dimensional Representations from Serial Sections," *J Microscopy, 96:* pp. 301-307, 1972.
11. Rakic, P., "Mode of Cell Migration to the Superficial Layers of Fetal Monkey Neocortex," *J Comp Neurol, 145:* pp. 61-84, 1972.
12. Macagno, R., V. Lopresti, C. Levinthal, "Structure and Development of Neuronal Connections in Isogenic Organisms: Variations and Similarities in the Optic System of *Daphnia magna*," *Proc Nat Acad Sci, 70:* pp. 57-61, 1973.
13. Rakic, P., L. J. Stensas, E. P. Sayre and R. L. Sidman, "Computer-aided Three-Dimensional Reconstruction and Quantitative Analysis of Cell from Serial Electron Microscopic Montages of Fetal Monkey Brain," *Nature, 250:* pp. 31-34, 1974.
14. Wiley, T. J., R. L. Schultz and A. H. Grott, "Computer Graphics in Three-Dimensions from Perspective Reconstruction of Brain Ultrastructure," *IEEE Trans Biomed Eng, 20:* pp. 288-291, 1973.
15. Cowan, W. M. and D. F. Wann, "A Computer System for the Measurement of Cell and Nuclear Sizes," *J Microscopy, 99:* pp. 331-348, 1973.
16. Levinthal, C. and R. Ware, "Three-Dimensional Reconstruction from Serial Sections," *Nature, 236:* pp. 207-210, 1972.
17. Wann, D. F., T. A. Woolsey, M. L. Dierker and W. M. Cowan, "An On-Line Digital-Computer System for the Semiautomatic Analysis of Golgi-Impregnated Neurons," *IEEE Trans Biomed Eng, 20:* pp. 233-247, 1973.
18. Glenn, W. V., R. J. Johnston, P. E. Morton and S. J. Dwyer, "Image Generation and Display Techniques for C.T. Scan Data," *Invest Radiol, 10:* pp. 403-416, 1975.
19. Glenn, W. V., R. J. Johnston, P. E. Morton and S. J. Dwyer, "Further Investigation and Initial Clinical Use of Advanced C.T. Display Capability," *Invest Radiol, 10:* pp. 479-489, 1975.
20. Huang, H. K. and R. S. Ledley, "Three-Dimensional Image Reconstruction from *in vivo* Consecutive Transverse Axial Sections," *Comput Biol Med, 5:* pp. 165-170, 1975.
21. Roberts, L. G., *Machine Perception of Three-Dimensional Solids*, Technical Report No. 315, Lincoln Laboratory, M.I.T., Cambridge, Mass., 1963.
22. Warnock, J., *A Hidden Surface Algorithm for Computer Generated Halftone Pictures*, Technical Report 4-15, Computer Science, University of Utah, Salt Lake City, Utah, 1969.
23. Sutherland, I. E., "Sketchpad, A Man-Machine Graphical Communication System," *Proc AFIPS, 23:* pp. 329-346, Spring, 1963.
24. Newman, W. M. and R. F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill Co., New York, 1973.
25. Bouknight, W. J., "A Procedure for Generation of Three-Dimensional Half-Tone Computer Graphic Presentations," *Comm Ass Comput Mach, 13:* pp. 527-536, 1970.
26. Bouknight, J. and K. Kelley, "An Algorithm for Producing Half-Tone Computer Graphics Presentations with Shadows and Movable Light Sources," *Proc AFIPS, 36:* pp. 1-10, Spring, 1970.
27. Watkins, G. S., *A Real-Time Visible Surface Algorithm*, Tech Report UTEC-CSC-70-101, Dept. Comput. Sci., University of Utah, Salt Lake City, Utah, 1969.
28. Wright, T. J., "A Two-Space Solution to the Hidden-Line Problem for Plotting Functions in Two Variables," *IEEE Trans Computers, 22:* pp. 28-33, 1973.
29. Golab, T., R. S. Ledley and L. S. Rotolo, "FIDAC-Film Input to Digital Automatic Computer," *Pattern Recog, 3:* pp. 123-156, 1971.
30. Ledley, R. S., "Pattern Recognition with Interactive Computing for a Half Dozen Clinical Applications of Health Care Delivery," *Proc AFIPS, 42:* pp. 463-477, 1973.
31. Golab, T. J., "MACDAC-An Inexpensive and Complete Biomedical Input and Output Display System," *Proc 23rd ACEMB*, Nov. 15-19, 1970.
32. Ledley, R. S., J. B. Wilson, T. Golab and L. S. Rotolo, "The ACTA-Scanner: The Whole Body Computerized Transaxial Tomograph," *Comput Biol Med, 4:* pp. 145-155, 1974.
33. Ledley, R. S., J. B. Wilson and H. K. Huang, "Reconstruction and Display of the Image from the ACTA Whole-Body X-Ray Scanner," *Proc 27th ACEMB*, Oct. 6-10, 1974.
34. Ledley, R. S. and L. S. Rotolo, "Advanced Features of the ACTA Scanner," *Proc 28th ACEMB*, Sept. 20-24, 1975.

35. Ledley, R. S., *Use of Computers in Biology and Medicine*, McGraw-Hill Co., New York, pp. 339-341, 1965.

36. Mazziotta, J. C., B. L. Hamilton and P. Fenner-Crisp, "A Device for the Precise Transfer of Serial Sections for Electron Microscopy," *Stain Tech, 48:* pp. 153-154, 1973.

37. Gouraud, H., "Continuous Shading of Curved Surfaces," *IEEE Trans on Computers, 20:* pp. 623-629, 1971.

38. Belson, M., A. W. Dudley and R. S. Ledley, "Automatic Computer Measurements of Neurons," *Pattern Recog, 1:* pp. 119-128, 1968.

Regional kidney transplant matching—The RENTRAN interactive approach*

by DAVID J. MISHELEVICH, PETER STASTNY, R. GAIL ELLIS and SUSAN G. MIZE
University of Texas Health Science Center at Dallas
Dallas, Texas

ABSTRACT

An interactive on-line computerized renal transplant matching system called RENTRAN which serves the Southwest Kidney Transplant Region is described. The region consists of one transplant center in Arkansas, two in Oklahoma, and six in Texas. The computer used is the DECsystem-10 located in the Medical Computing Resources Center at the University of Texas Health Science Center at Dallas. RENTRAN participants have remotely located standard interactive computer terminals and gain access to the computer by dialing over normal telephone lines. Functions provided by RENTRAN include obtaining instructions, performing a donor-recipient match, obtaining a list of potential recipients, making a user comment or adding, updating or deleting a patient record. Either long or short dialog forms are available for inexperienced and experienced users respectively. In the time period since August 1, 1973 when the system went into production, there have been in excess of 100 matches attempted and approximately 50 kidneys have been transplanted according to RENTRAN results. There are about 350 recipients currently on the data base. The system was developed with funds provided by the Texas Regional Medical Program and operational expenses and enhancements are provided through a $25 per year potential-recipient patient charge for being listed on the data base.

INTRODUCTION

Success in renal transplant matching can be increased by enlarging the pools of potential recipients and donors. With this in mind, the Southwest Kidney Transplant Region maintains an interactive on-line computer-based matching system called RENTRAN in collaboration with the Medical Computing Resources Center (MCRC) of The University of Texas Health Science Center at Dallas (UTHSCD).

The network is shown schematically in Figure 1 and includes one center in Arkansas (Little Rock), two centers in Oklahoma (V.A. Hospital in Oklahoma City and St. Anthony's Hospital) and six centers in Texas (Austin, Dallas, Galveston, Houston and two in San Antonio). The latter two are located at Lackland Air Force Base and The University of Texas Health Science Center at San Antonio. The three states comprise the Southwest Kidney Transplant Region and maintain common tissue typing standards including the holding of typing workshops, use of a common tissue typing tray and exchange of sera from sensitized recipients for cross-matching.

This paper describes system capabilities, including sample computer terminal interactions. The results for the first year and one half of operation are presented

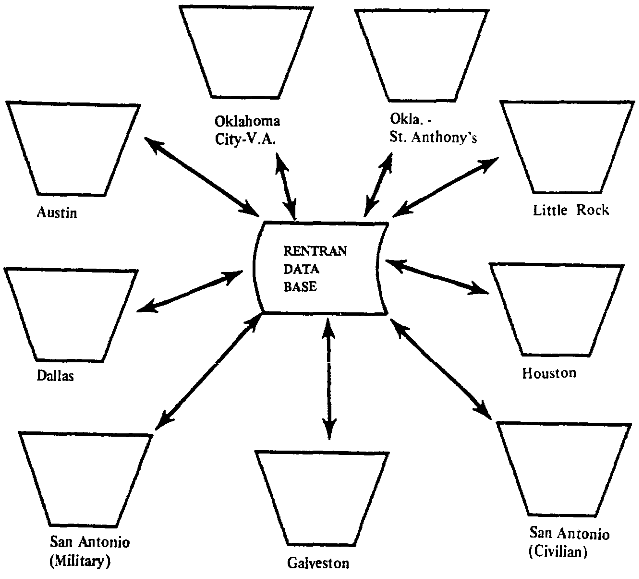


Figure 1—Overall threetate RENTRAN network

* The major portion of financial support for RENTRAN came from the Renal Program, Texas Regional Medical Program.

and a comparison of features with the SEROP system[1] is made.

SYSTEM DESCRIPTION

The interactive on-line computerized system replaced a previous batch system whose major defects were relatively infrequent updating (monthly) of potential recipient data and the necessity for human determination of matches based on scanning lists of potential recipients comparing the blood types and up to four HL-A values. The current system went into production on August 1, 1973 with development occurring earlier that year. A prototypical demonstration version was programmed in the BASIC language in two months, beginning in December 1972. The production version of the RENTRAN program was begun in March 1973 in the COBOL computer language using the ISAM (Indexed Sequential Access Method) file structure and runs on the DECsystem-10 computer located in the MCRC. The MCRC is an academic and scientific computing facility that serves some administrative functions as well.[2]

A prime objective in the system design was to put the computing power where the users are. The computing system has a number of dial-up lines and supports a variety of interactive computer terminals. The transplant centers in the three-state region communicate with the MCRC DECsystem-10 over normal telephone lines at rates of 10 or 30 characters per second. The terminals are equipped with acoustic couplers for digital to analog sound signal conversion and vice versa to permit the communication. Data confidentiality is maintained by the use of account numbers and passwords on the time sharing system.

The RENTRAN data base is a file of potential kidney transplant recipients currently numbering about 350. The following items of information are kept for each patient:

Registry number
Name
Age
Sex
Race
Blood Type (ABO)
HL-A Antigens
Location Center
Source of Reference
Presence or absence of antibodies
Number of acceptable mismatched antigens.

This information is maintained daily on disk storage in the computer system and on back-up magnetic tapes. In addition, monthly printouts are made of the file for comparative and back-up purposes.

The user has a choice of the following functions:

Obtain instructions
Perform a donor-recipient match
List potential recipients
Make a user comment
Add, update, or delete a patient.

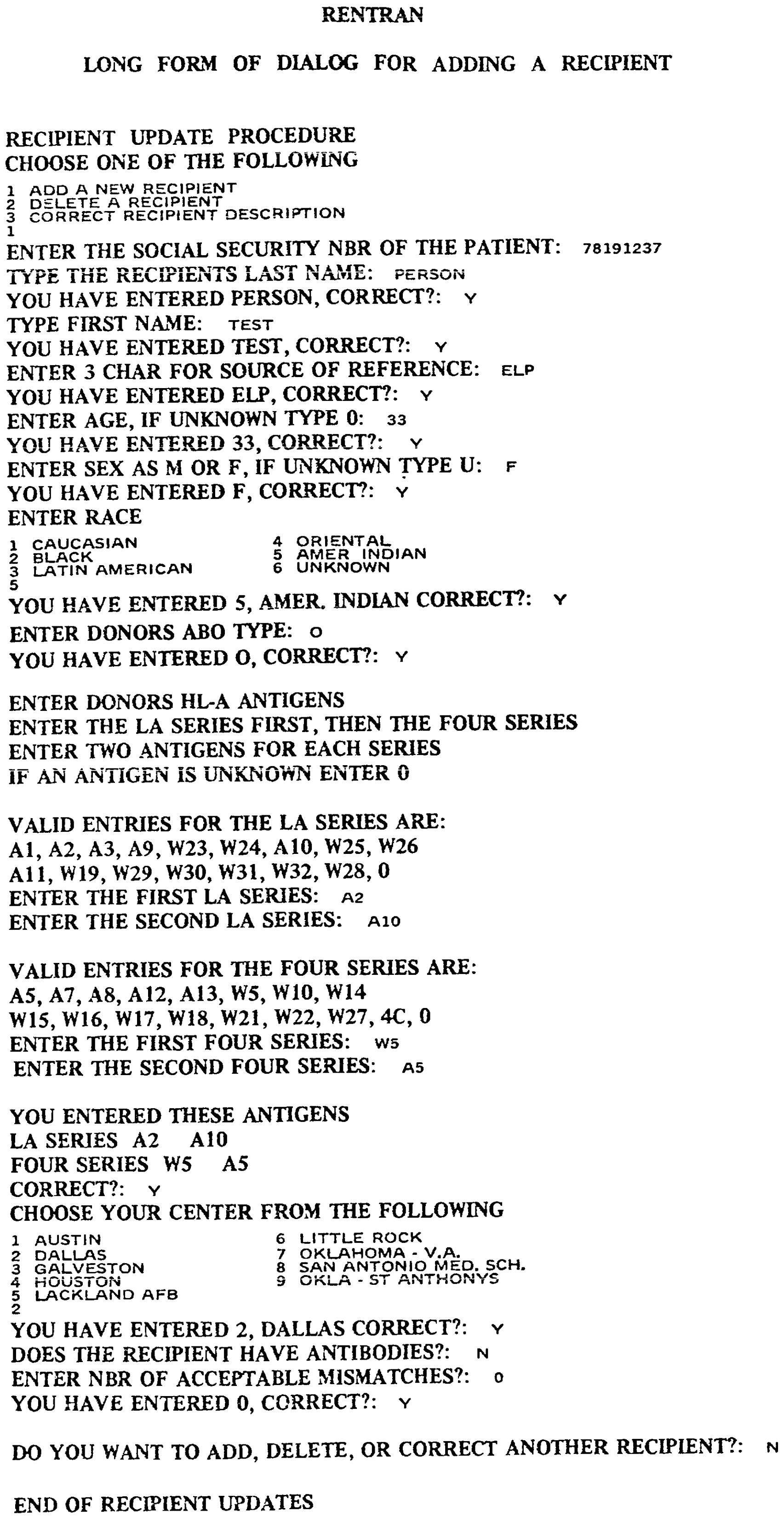
Since the system is designed to minimize typing at the computer terminal, the choices for these and other multiple choice items are numbered. Thus the user simply has to type in the appropriate number and press the carriage return to enter the selection into the computer. A sample update procedure demonstrating the addition of a patient is shown in Figure 2. This interaction was human engineered to make RENTRAN easy to use by an individual without computer experience. Each user entry is edited for consistency and the user is obliged to enter a meaningful response before proceeding to the next entry. Because more experienced users found this dialog with its many checks to be long and tedious, a short dialog version was developed. The same process in short-dialog form is shown in Figure 3. In this format if the user forgets the possible values in a multiple choice question, he or she can type in a 0 (zero) followed by a carriage return and the possibilities will be typed out to prompt the user. Editing for consistency also applies to the short dialog.

Updates are actually stored in a temporary file and once a day the added, changed, or deleted data is checked for consistency and processed by a maintenance program to bring the master RENTRAN file up to date. This indirect approach was selected to protect data integrity by avoiding problems with users of varied experience making changes directly to the master data bank.

A sample patient listing with the names removed appears in Figure 4. The primary listing choices are all patients or those within a given center. Once a specific center is designated, the variety of lists that may be obtained for that center includes the following:

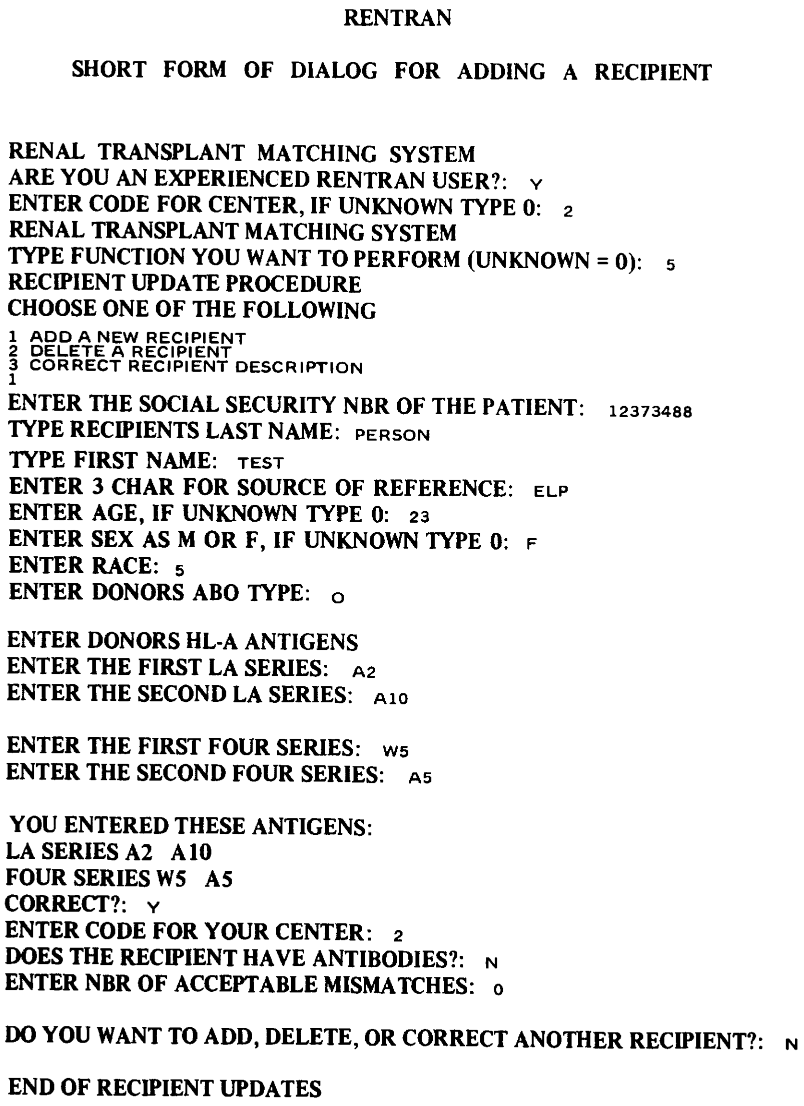
All potential recipients having a certain blood type
All those with a specified number of HL-A antigens
All those with a certain number of acceptable mismatches
All those who have developed antibodies
All those with a specific source of reference
All patients for the center listed alphabetically
All patients for the center in registry number sequence.

Two additional listing capabilities which are independent of the center are obtaining the data for a single patient provided the registry number is known and listing those patients who have been added as potential recipients during the current month.



```
                         RENTRAN

          LONG FORM OF DIALOG FOR ADDING A RECIPIENT


RECIPIENT UPDATE PROCEDURE
CHOOSE ONE OF THE FOLLOWING
1 ADD A NEW RECIPIENT
2 DELETE A RECIPIENT
3 CORRECT RECIPIENT DESCRIPTION
1
ENTER THE SOCIAL SECURITY NBR OF THE PATIENT: 78191237
TYPE THE RECIPIENTS LAST NAME: PERSON
YOU HAVE ENTERED PERSON, CORRECT?: Y
TYPE FIRST NAME: TEST
YOU HAVE ENTERED TEST, CORRECT?: Y
ENTER 3 CHAR FOR SOURCE OF REFERENCE: ELP
YOU HAVE ENTERED ELP, CORRECT?: Y
ENTER AGE, IF UNKNOWN TYPE 0: 33
YOU HAVE ENTERED 33, CORRECT?: Y
ENTER SEX AS M OR F, IF UNKNOWN TYPE U: F
YOU HAVE ENTERED F, CORRECT?: Y
ENTER RACE
1 CAUCASIAN            4 ORIENTAL
2 BLACK                5 AMER INDIAN
3 LATIN AMERICAN       6 UNKNOWN
5
YOU HAVE ENTERED 5, AMER. INDIAN CORRECT?: Y
ENTER DONORS ABO TYPE: O
YOU HAVE ENTERED O, CORRECT?: Y

ENTER DONORS HL-A ANTIGENS
ENTER THE LA SERIES FIRST, THEN THE FOUR SERIES
ENTER TWO ANTIGENS FOR EACH SERIES
IF AN ANTIGEN IS UNKNOWN ENTER 0

VALID ENTRIES FOR THE LA SERIES ARE:
A1, A2, A3, A9, W23, W24, A10, W25, W26
A11, W19, W29, W30, W31, W32, W28, 0
ENTER THE FIRST LA SERIES: A2
ENTER THE SECOND LA SERIES: A10

VALID ENTRIES FOR THE FOUR SERIES ARE:
A5, A7, A8, A12, A13, W5, W10, W14
W15, W16, W17, W18, W21, W22, W27, 4C, 0
ENTER THE FIRST FOUR SERIES: W5
ENTER THE SECOND FOUR SERIES: A5

YOU ENTERED THESE ANTIGENS
LA SERIES A2    A10
FOUR SERIES W5    A5
CORRECT?: Y
CHOOSE YOUR CENTER FROM THE FOLLOWING
1 AUSTIN               6 LITTLE ROCK
2 DALLAS               7 OKLAHOMA - V.A.
3 GALVESTON            8 SAN ANTONIO MED. SCH.
4 HOUSTON              9 OKLA - ST ANTHONYS
5 LACKLAND AFB
2
YOU HAVE ENTERED 2, DALLAS CORRECT?: Y
DOES THE RECIPIENT HAVE ANTIBODIES?: N
ENTER NBR OF ACCEPTABLE MISMATCHES?: 0
YOU HAVE ENTERED 0, CORRECT?: Y

DO YOU WANT TO ADD, DELETE, OR CORRECT ANOTHER RECIPIENT?: N

END OF RECIPIENT UPDATES
```

Figure 2—Sample update procedure demonstrating the addition of a patient using the long dialog. Stylized type used for emphasis as well as to differentiate computer output and user responses



```
                              RENTRAN

           SHORT FORM OF DIALOG FOR ADDING A RECIPIENT


RENAL TRANSPLANT MATCHING SYSTEM
ARE YOU AN EXPERIENCED RENTRAN USER?: Y
ENTER CODE FOR CENTER, IF UNKNOWN TYPE 0: 2
RENAL TRANSPLANT MATCHING SYSTEM
TYPE FUNCTION YOU WANT TO PERFORM (UNKNOWN = 0): 5
RECIPIENT UPDATE PROCEDURE
CHOOSE ONE OF THE FOLLOWING
1 ADD A NEW RECIPIENT
2 DELETE A RECIPIENT
3 CORRECT RECIPIENT DESCRIPTION
1
ENTER THE SOCIAL SECURITY NBR OF THE PATIENT: 12373488
TYPE RECIPIENTS LAST NAME: PERSON
TYPE FIRST NAME: TEST
ENTER 3 CHAR FOR SOURCE OF REFERENCE: ELP
ENTER AGE, IF UNKNOWN TYPE 0: 23
ENTER SEX AS M OR F, IF UNKNOWN TYPE 0: F
ENTER RACE: 5
ENTER DONORS ABO TYPE: O

ENTER DONORS HL-A ANTIGENS
ENTER THE FIRST LA SERIES: A2
ENTER THE SECOND LA SERIES: A10

ENTER THE FIRST FOUR SERIES: W5
ENTER THE SECOND FOUR SERIES: A5

YOU ENTERED THESE ANTIGENS:
LA SERIES A2  A10
FOUR SERIES W5  A5
CORRECT?: Y
ENTER CODE FOR YOUR CENTER: 2
DOES THE RECIPIENT HAVE ANTIBODIES?: N
ENTER NBR OF ACCEPTABLE MISMATCHES: 0

DO YOU WANT TO ADD, DELETE, OR CORRECT ANOTHER RECIPIENT?: N

END OF RECIPIENT UPDATES
```

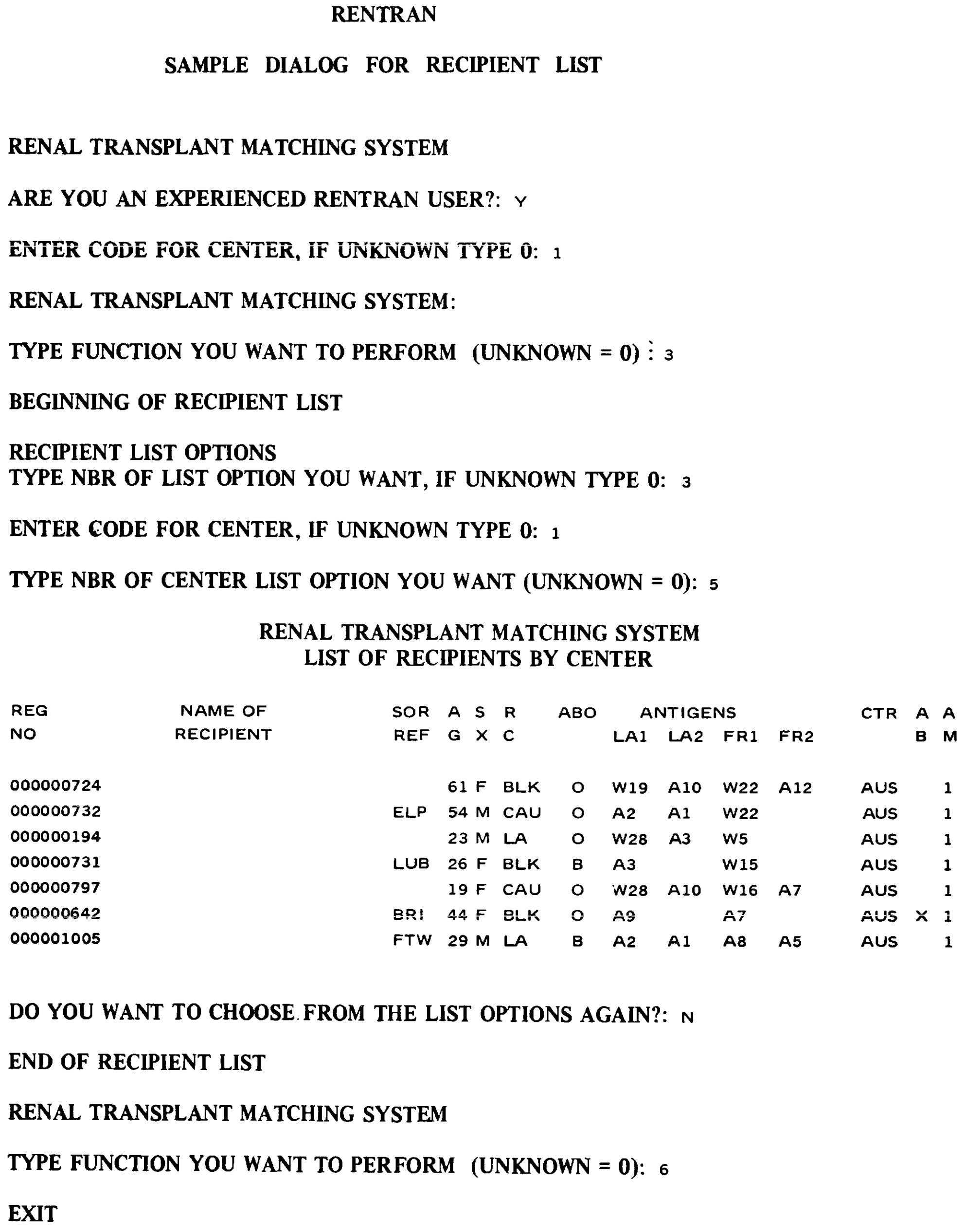
Figure 3—Sample update procedure demonstrating the addition of a patient using the short dialog

The dialog associated with a matching procedure is shown in Figure 5. It is important to note that cross-matching of antigens is permitted. For example antigen designation 4C will show matches to A5, W5, and W18. Each time a match is performed, a follow-up form is mailed to the donor center to determine the disposition of the kidneys.

The user of RENTRAN can get minimal instructions for system use on the terminal even though full user documentation is supplied to each center. This permits the system to be usable even if the documentation is mislaid. A telephone reference is given for special problems that might arise.

An important capability is the user comment facility. Problems, general questions or other items of interest are input by the users and reviewed on a daily basis when the updating of the master file is done. Changes may result from these comments, such as the provision of an abbreviated as well as a long form of the dialog to expedite interaction by the experienced users.

The DECsystem-10 is available 24 hours per day, 7 days per week except for an average of 2½ hours of preventive maintenance and rare episodes of unscheduled down time. In almost two years there has been no time when a match could not be done even if some delay was necessary. Periodically a printed listing of all potential recipients on the data base is distributed to each member center to be used for backup in case of emergencies. It is still, however, the objective of



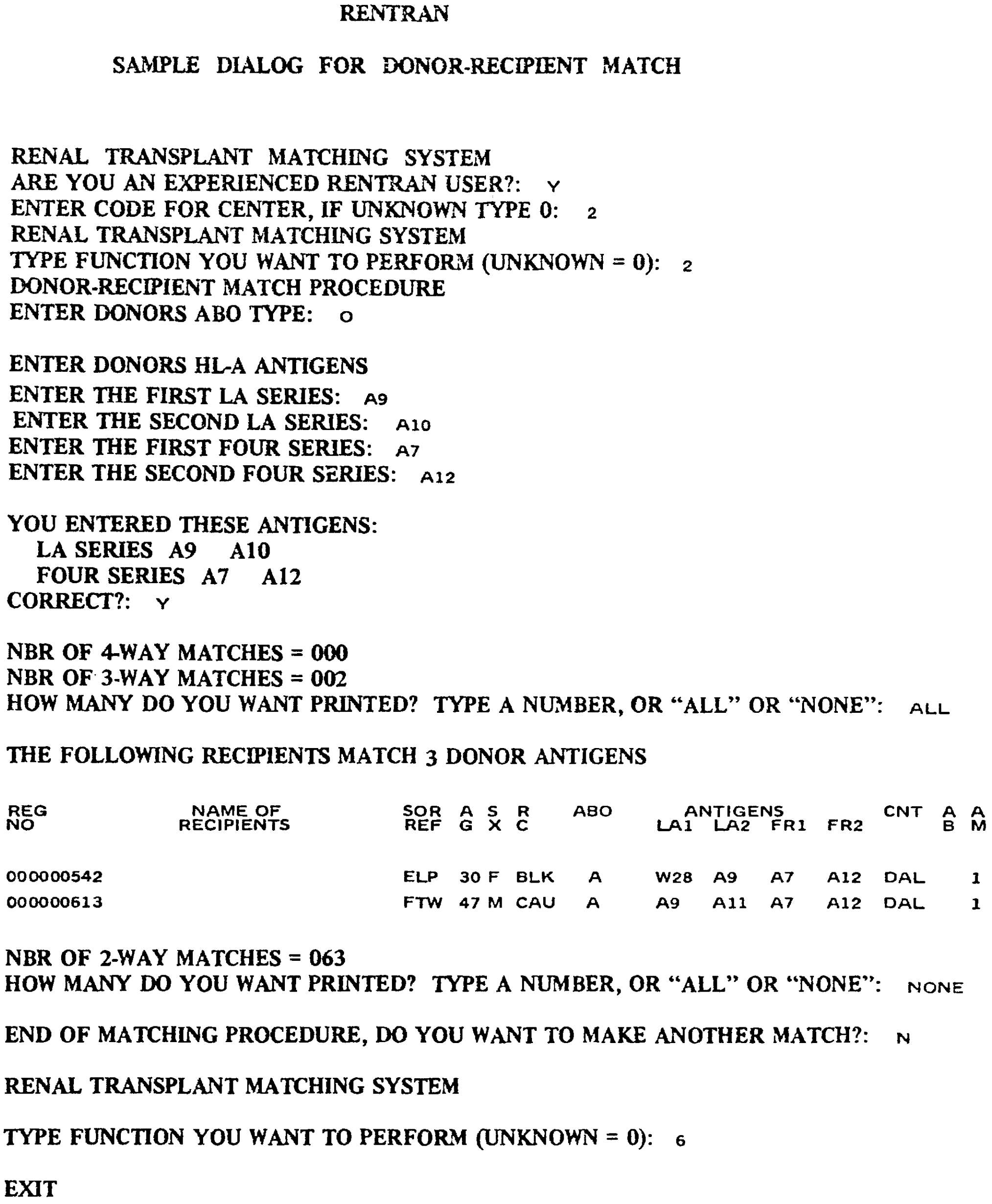
```
                         RENTRAN

           SAMPLE DIALOG FOR RECIPIENT LIST


RENAL TRANSPLANT MATCHING SYSTEM

ARE YOU AN EXPERIENCED RENTRAN USER?: Y

ENTER CODE FOR CENTER, IF UNKNOWN TYPE 0: 1

RENAL TRANSPLANT MATCHING SYSTEM:

TYPE FUNCTION YOU WANT TO PERFORM (UNKNOWN = 0) : 3

BEGINNING OF RECIPIENT LIST

RECIPIENT LIST OPTIONS
TYPE NBR OF LIST OPTION YOU WANT, IF UNKNOWN TYPE 0: 3

ENTER CODE FOR CENTER, IF UNKNOWN TYPE 0: 1

TYPE NBR OF CENTER LIST OPTION YOU WANT (UNKNOWN = 0): 5

                 RENAL TRANSPLANT MATCHING SYSTEM
                   LIST OF RECIPIENTS BY CENTER

REG          NAME OF        SOR  A  S R    ABO    ANTIGENS           CTR A A
NO           RECIPIENT      REF  G  X C          LA1 LA2 FR1 FR2         B M

000000724                        61 F BLK   O    W19 A10 W22 A12     AUS   1
000000732                   ELP  54 M CAU   O    A2  A1  W22         AUS   1
000000194                        23 M LA    O    W28 A3  W5          AUS   1
000000731                   LUB  26 F BLK   B    A3      W15         AUS   1
000000797                        19 F CAU   O    W28 A10 W16 A7      AUS   1
000000642                   BRI  44 F BLK   O    A9      A7          AUS X 1
000001005                   FTW  29 M LA    B    A2  A1  A8  A5      AUS   1

DO YOU WANT TO CHOOSE FROM THE LIST OPTIONS AGAIN?: N

END OF RECIPIENT LIST

RENAL TRANSPLANT MATCHING SYSTEM

TYPE FUNCTION YOU WANT TO PERFORM (UNKNOWN = 0): 6

EXIT
```

Figure 4—Sample listing, in this case by center, although a list of all patients could be requested. Patient names have been removed to preserve confidentiality

the MCRC to develop a fully redundant fail-safe system to further increase availability for such critical activities.

RESULTS

Some basic statistics of RENTRAN since initiation of the production system on August 1, 1973 are demonstrated in Table I. As of March 1, 1975 there were 356 potential recipients on the file and 108 donor-recipient matches have been performed. The data gathered by analysis of the follow-up forms mailed after each attempted match or deletion of a recipient from the data base are shown in Table II. In some cases a kidney was transported to an area outside the regional membership. RENTRAN results were used in over 50 cases to identify the appropriate recipient.



```
                        RENTRAN

          SAMPLE DIALOG FOR DONOR-RECIPIENT MATCH


RENAL TRANSPLANT MATCHING SYSTEM
ARE YOU AN EXPERIENCED RENTRAN USER?: Y
ENTER CODE FOR CENTER, IF UNKNOWN TYPE 0: 2
RENAL TRANSPLANT MATCHING SYSTEM
TYPE FUNCTION YOU WANT TO PERFORM (UNKNOWN = 0): 2
DONOR-RECIPIENT MATCH PROCEDURE
ENTER DONORS ABO TYPE: O

ENTER DONORS HL-A ANTIGENS
ENTER THE FIRST LA SERIES: A9
ENTER THE SECOND LA SERIES: A10
ENTER THE FIRST FOUR SERIES: A7
ENTER THE SECOND FOUR SERIES: A12

YOU ENTERED THESE ANTIGENS:
  LA SERIES A9   A10
  FOUR SERIES A7   A12
CORRECT?: Y

NBR OF 4-WAY MATCHES = 000
NBR OF 3-WAY MATCHES = 002
HOW MANY DO YOU WANT PRINTED? TYPE A NUMBER, OR "ALL" OR "NONE": ALL

THE FOLLOWING RECIPIENTS MATCH 3 DONOR ANTIGENS

REG             NAME OF           SOR  A  S  R    ABO      ANTIGENS         CNT  A  A
NO             RECIPIENTS         REF  G  X  C          LA1  LA2  FR1  FR2       B  M

000000542                         ELP  30 F  BLK   A    W28  A9   A7   A12  DAL     1
000000613                         FTW  47 M  CAU   A    A9   A11  A7   A12  DAL     1

NBR OF 2-WAY MATCHES = 063
HOW MANY DO YOU WANT PRINTED? TYPE A NUMBER, OR "ALL" OR "NONE": NONE

END OF MATCHING PROCEDURE, DO YOU WANT TO MAKE ANOTHER MATCH?: N

RENAL TRANSPLANT MATCHING SYSTEM

TYPE FUNCTION YOU WANT TO PERFORM (UNKNOWN = 0): 6

EXIT
```

Figure 5—Interactive dialog demonstrating the matching procedure

Another area of interest involves reasons for the permanent or temporary removal of potential recipients from the data base. The information is shown in Table III. There were 263 permanent removals (142 because of transplantation, 51 because of death, and 70 because of miscellaneous reasons as analyzed in the figure) and 26 temporary removals (of which 5 were due to infection).

FINANCIAL ASPECTS

The development of RENTRAN began in December of 1972. The initial development cost was approximately $9,000 with roughly four-fifths coming from the Texas Regional Medical Program. The computer terminal and telephone costs are borne by the individual transplantation centers. Operating expenses and

TABLE I—Basic RENTRAN statistics as of March 1, 1975. The period covered is July, 1973 through February, 1975.

RENAL TRANSPLANT MATCHING SYSTEM (RENTRAN)

STATISTICAL ANALYSIS

AS OF 03-01-75

			TOTALS			
DESCRIPTION	02-01-74 TO 03-01-75		07-01-73 TO 02-01-74		NUMBER	
DONOR-RECIPIENT MATCHES PERFORMED:						
KIDNEY AVAILABLE FOR MATCHING	54		17		71	
NO INFORMATION AVAILABLE	29		8		37	
TOTAL		83		25		108
REQUESTS FOR RECIPIENT LISTINGS		120		45		165
UPDATES TO THE RECIPIENT FILE						
RECIPIENTS ADDED	530		174		704	
RECIPIENTS DELETED	192		145		337	
RECIPIENTS CORRECTED	195		148		343	
TOTAL		917		467		1384
INSTRUCTIONS REQUESTED		39		7		46
NUMBER OF SUCCESSFUL SIGN-ONS		287		129		416
RECIPIENTS ON THE DATA BASE (AS OF ENDING PERIOD)		356		245		356

TABLE II—Data obtained from RENTRAN follow-up forms.

RENTRAN

ANALYSIS OF FOLLOW-UP DATA

			TOTALS			
DESCRIPTION	02-01-74 TO 03-01-75		07-01-73 TO 02-01-74		NUMBER	
FOLLOW-UP INFORMATION						
FORMS FOR RECIPIENTS DELETED FROM DATA BASE						
RETURNED	181		108		289	
NOT YET RETURNED	11		37		48	
TOTAL		192		145		337
FORMS FOR DONOR-RECIPIENT MATCH ATTEMPTS						
RETURNED	54		17		71	
NOT RETURNED	29		8		37	
TOTAL		83		25		108
REPORTED TRANSPLANTS OF KIDNEYS WHERE DONOR WAS MATCHED TO RECIPIENT USING RENTRAN (BY RECIPIENT LOCATION CENTER)						
HOUSTON	3		9		12	
GALVESTON	4		3		7	
DALLAS	24		2		26	
LACKLAND			2		2	
AUSTIN			1		1	
LITTLE ROCK						
OKLAHOMA CITY-UNIV OF OKLA.	4		2		6	
SAN ANTONIO	3		1		4	
NEW YORK	2				2	
OKLAHOMA-ST ANTHONY'S	4				4	
NASHVILLE, TENN.			1		1	
KIDNEY NOT USED	47		8		55	
DISPOSITION OF KIDNEY UNKNOWN			5		5	
TOTAL		91		34		125

TABLE III—Reasons for the permanent or temporary removal of potential recipients from the data base.

RENTRAN

REASONS FOR REMOVAL OF RECIPIENTS FROM DATA BASE

					TOTALS				
DESCRIPTION	02-01-74 TO 03-01-75			07-01-73 TO 02-01-74			NUMBER		
REPORTED REASONS FOR REMOVAL OF POTENTIAL RECIPIENT FROM DATA BASE:									
PERMANENT REMOVAL FROM DATA BASE DUE TO:									
TRANSPLANT		90			52			142	
DEATH		38			13			51	
OTHER									
NOT A CANDIDATE FOR TRANSPLANT	26			7			33		
HAS A LIVING RELATED DONOR	5			6			11		
REFUSAL BY PATIENT	6			2			8		
UNABLE TO LOCATE RECIPIENT	1			1			2		
MOVED AWAY FROM AREA	3			1			4		
ERROR INPUT	2						2		
ERROR-ENTERED TWICE ·ON RENTRAN	4			3			7		
FORM RETURNED BUT NOT FILLED OUT	1			2			3		
TOTAL OTHER		48			22			70	
TOTAL			176			87			263
TEMPORARY REMOVAL FROM DATA BASE DUE TO:									
INFECTION					5			5	
OTHER									
NOT READY	5			16			21		
TOTAL OTHER		5			16			21	
TOTAL			5			21			26

periodic RENTRAN enhancements are provided by a $25 charge made once during a September to August fiscal year to each potential recipient listed on the file irrespective of the length of time on the file during the year. Patients are not charged twice if they are temporarily removed and then reinstated during the same fiscal year. Patients are individually billed through their respective centers for RENTRAN via a computerized billing system.

DISCUSSION

The system is much more successful than the preceding batch system. The concept of using a centralized up-to-date file with access over telephone lines has proven to be quite reasonable. There are other such computerized systems in existence, a major one being the SEROP (Southeastern Regional Organ Procurement Program) system.[1] A detailed comparison of its characteristics with those of RENTRAN is shown in Table IV. SEROP currently covers a larger area with its 30 participating institutions. While there is no practical limit to the number of centers or potential recipients which could be supported by RENTRAN, the Southwest Kidney Transplant Region has developed on the basis of geographic proximity and cooperation among the physicians involved. It must be stressed that most of the decisions are medical ones and therefore are made by the medical group which oversees RENTRAN activities. A somewhat more mathematical approach to computerized donor-recipient matching has been developed by the Department of Immunohematology, University Hopital, Leyden, the Netherlands.[3] A key to the success of RENTRAN and a main objective in its design was to make the system easy to use for those trained or untrained in its use.

CONCLUSION

Renal transplantation is the most successful of all internal organ transplantation procedures. Pooling of recipients in regional exchange groups allows more efficient utilization of available cadaver kidneys. The computer-based regional approach such as that of RENTRAN which provides an accurate and up-to-date file of a fairly large size to maximize the probability of finding a likely potential recipient and which is accessible via telephone lines from essentially any distance has proved to be a viable one. The RENTRAN system, which is readily available and easily used, is successfully utilized by the participating transplant centers.

TABLE IV—Comparison of RENTRAN features and information to those of the system supported by SEROP.

A COMPARISON OF RENTRAN TO THE SEROP TRANSPLANT SYSTEM

SYSTEM FEATURE	AS USED BY SEROP SYSTEM	AS USED BY RENTRAN
Recipient Identification	includes # of transplants, RH factor, antigens against which patient is sensitized	social security #, source of referral within center
Donor Identification	Basic ID - Name, age, location, etc.	None
Update Technique	Direct to files. Brief & succinct but requires training in format program expects.	Indirectly, once per day for file protection. Verbose or abbreviated form user's choice. Training not necessary.
# programs	Separate program for each function (update, list, etc.)	One user program combining functions
Patient protection within center	Password method	None
Matching criteria	(1) HL-A compatibility (2) presensitization (3) waiting time	HL-A compatibility
Billing capability	None	Basic monthly audit trail
24-hour availability	Network of 8 GE Mark III's available. Maximum of 30 minutes downtime guaranteed	When possible
Instructions for use	None programmed	Programmed with telephone reference
User comment facility?	No	Yes
Follow-up after match	None described	By mailout
Cross-matching of antigens	None	Extensive
Statistics	3½ years implementation 6 major revisions 800 recipients on file 30 participating institutions 10 states	2 years implementation 1 major revision 356 recipients 9 participating institutions 3 states
Cost	R & D = $20,000 over 4½ yrs. maintenance = $550 per month for 800 patients	R & D = $9,000 over 1 year Maintenance and enhancements $25 per patient per year

ACKNOWLEDGMENTS

Thanks are gratefully given to the members of the Southwest Kidney Transplant Region, the Texas Regional Medical Program, Dr. Garabed Eknoyan of the Baylor College of Medicine who headed the renal section of the Texas RMP, Mr. Stephen B. Brewton who programmed the original prototypical demonstration program in BASIC, Ms. Linda Firnberg who participated in the systems analysis and training, and Mr. Michael Blaylock who programmed many of the later changes to RENTRAN.

REFERENCES

1. Stulting, R. D. and F. E. Ward, "A Computerized System for the Selection of Organ Transplant Recipients," *Transplantation*, 19, pp. 27-35, 1975.
2. Mishelevich, D. J., "Medical Computing in Texas," *Medical Electronics and Data*, 6, pp. 33-37, 1975.
3. Steen, G. J. and J. D'Amaro, "Computer Selection of Donor-Recipient Pairs," *Tissue Antigens*, 2, pp. 189-95, 1972.

Data base for protein sequences*

by M. O. DAYHOFF, W. C. BARKER, R. M. SCHWARTZ,
B. C. ORCUTT, and L. T. HUNT
National Biomedical Research Foundation
Washington, D.C.

ABSTRACT

Proteins are linear polymers synthesized in living organisms from twenty different kinds of amino acids according to the message carried in the chromosomes. Typically, they have evolved by natural selection over hundreds of millions of years through many small changes in sequence. The present computerized data base includes 77,267 amino acid residues from 767 sequences. We believe that all of the protein structures occurring in living organisms can be combined into fewer than 1,000 groups containing proteins of similar sequence. Each group can be characterized by a few sequences that are known exactly and by a number of evolutionary parameters. The rest of the structures, occurring in organisms not examined, can be described with estimated precision in terms of the number of differences from a known sequence or from sequences inferred to have been present in ancestral forms. The following kinds of information are needed: sequences of proteins from each group, the phylogenetic tree of biological species, a list of protein groups and the gene duplications in each, a description of the quantitative parameters of the evolutionary processes affecting proteins, and methods of estimation of sequences in ancestral species and in living forms phylogenetically close to those investigated. The conceptual tools and the computer programs necessary for the prediction of all of the 10^{10} to 10^{11} protein sequences in living species are described. One can readily visualize the separate parts operating as an integrated interactive computerized data base that could predict sequences for specified organisms with an estimated precision based on the collection of known sequences.

Proteins are the most interesting of all of the biochemical components of living organisms because of their potential for diversity in structure and function, their key positions in the cellular fabric, catalysis, and control mechanisms, and their great conservation of structure, which provides information about biological evolution. Sequence studies have been carried out for various reasons in a number of disciplines: structural chemistry, crystallography, enzymology, genetics, evolution, systematics, physiology, immunology, pharmacology, virology, and bacteriology, to name a few. In many of these disciplines, the protein sequences and associated data are essential for an understanding of the fundamental concepts and for a parsimonious organization of the knowledge.

Proteins are synthesized by complex cell machinery according to the information in the deoxyribonucleic acid (DNA) template, which is replicated in cell division or reproduction. There are 4 kinds of nucleotides in the linear DNA chain and there are 20 kinds of amino acids in the linear chain of the protein. Some proteins are functional as initially polymerized; some undergo subsequent chemical modification.

Occasional errors are made in the copying of the DNA from generation to generation. When such errors are beneficial or neutral, the new sequence may eventually replace the original as the predominant form in the species. The main kinds of errors reflected in the protein sequences are point mutations (the exchange of one amino acid for another) and insertions or deletions of one or a few residues in the chain. Although errors occur in many individuals each generation, it may be millions of years before a change in any one protein permeates the whole population.

PRESENT AND ULTIMATE SIZE OF THE PROTEIN SEQUENCE DATA COLLECTION

We maintain a reference data collection of protein sequences embodied in a series of books entitled *Atlas of Protein Sequence and Structure*,[1,2,3] in computer tapes containing the sequences,[4] and in derivative data such as a key-amino-acid-in-context dictionary.[5] Over 3,000 experimental scientists have elucidated the 77,267 amino acid residues in the 767 sequences currently on our sequence data tape. These sequences represent only a small sample of the entire data base that could

* This work has been supported by NIH Grant GM-08710 from the National Institute of General Medical Sciences and by Contract NASW2546 from the National Aeronautics and Space Administration.

be compiled from the 10^{10} or 10^{11} different protein sequences from the approximately 2 million living species; in humans alone between 10^5 and 10^6 different proteins are thought to be expressed.

We believe that it will be possible to combine all of the naturally occurring protein structures into fewer than 1,000 groups containing proteins of similar sequence.[6,7] Each group (or superfamily) can be characterized by a sampling of sequences that are known exactly and by a number of evolutionary parameters. The rest of the structures, occurring in organisms not examined, can be described with estimated precision in terms of the number of differences from a known sequence or from sequences inferred to have been present in ancestral forms on the basis of the data from extant species. Thus, virtually the entire collection of proteins of living organisms can be described with useful precision by the elucidation of only a limited number of sequences and parameters for each group.

Procedures for organizing and estimating the chemical structures of proteins in living organisms require the following kinds of information:

1. The phylogenetic tree describing the evolution of extant biological species
2. The organization of sequences into superfamilies
3. A tabulation of the gene duplications that have occurred in each superfamily
4. The frequency of occurrence of point mutations and insertion-deletion events for each superfamily
5. A method for the estimation of protein sequences that occurred in ancestral species from those found in living organisms
6. A method for the estimation of a protein sequence in a living organism based on related sequences in other organisms and on inferred ancestral sequences

THE PHYLOGENETIC TREE OF LIFE

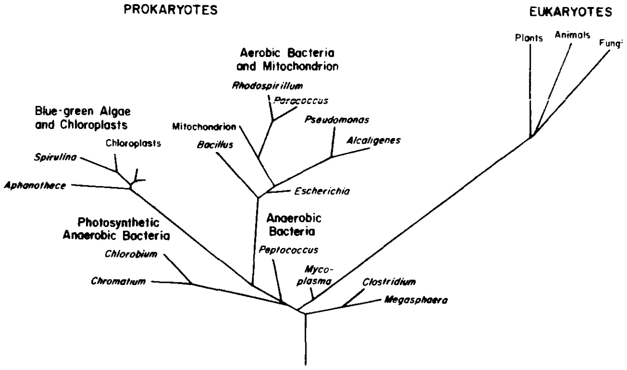


Figure 1—Phylogenetic tree of living organisms. The major groups of bacteria and blue-green algae as well as the eukaryote host and its endosymbionts are represented. Spirochaetes that may be the endosymbiotic ancestors of flagella and the mitotic apparatus are not pictured, as no sequence information from them is yet available. The topology and branch lengths were approximated from sequences of c-type cytochromes, ferredoxins, plastocyanin and azurin, and 5S ribosomal RNA. The position of *Mycoplasma*, estimated from transfer RNA sequences, is only tentative. The lengths of the branches reflect the amount of change that has taken place in the sequences of the various organisms

From fossil evidence and from biological and protein sequence evidence[3,8–11] an outline of the phylogenetic tree of life is emerging, as shown in Figure 1. About 2.8 billion years ago there were at least two morphologically distinct types of organisms already living on earth: one was filamentous and the other spherical. Presumably the genetic code and a rudimentary metabolism had already developed. Highly conserved descendants of these forms may be the filamentous and coccoid blue-green algae and the anaerobic bacteria such as clostridia. About 2.0 billion years ago, when an oxidizing atmosphere began to develop, a large number of different morphological types of prokaryotes abounded, presumably including the beginnings of the rest of the 19 major groups of bacteria found today as well as a proliferation of blue-green algal types. It is thought that one of the bacterial lines, possibly mycoplasma, gave rise to the host cell of the higher organisms, or eukaryotes. Another bacterial line invaded the host, developed a symbiotic relationship, and gave rise to the mitochondrion, in which the oxidative reactions of the cell take place. By a similar mechanism, blue-green algae gave rise to plant chloroplasts, in which photosynthesis takes place. The host cell and its endosymbiont(s) have evolved together in most of the multicellular forms.

About 1 billion years ago the lines that have since evolved into the present-day fungi, plants, and animals diverged from each other (see Figure 2).[12–16] Proceeding along the animal line leading to human, the insect line diverged first, followed by the annelid and mollusc lines and then, about 400 million years ago, by the line to the fishes. More recently the amphibians branched off, then the reptiles and birds. Finally, about 75 million years ago there was a rapid proliferation of mammalian species into some 19 orders. Among the primates the line to monkeys separated early, followed by the divergence of the great apes from the human line.[17] On many of the other branches of the tree there has been a comparable proliferation of groups. It seems reasonable to suppose that the major branches, representing biological orders, classes, and phyla, along with their approximate times of divergence and average amounts of evolutionary change, will soon be fitted into place, largely on the basis of sequence evidence. Much is already known from morphological evidence about the connections of species, genera, and families in the terminal portions of all of the branches, which represent the 2 million living species. At present it suffices to consider in our tree the few hundred species for which we have sequences and the approximately one thousand intermediate groups, such as families, about which most inquiries would be made.

The topology of this tree, the names of the species or groups on all of the branch tips, and the lengths of the branches, which reflect average evolutionary change of the proteins in the species, can all be stored in the computer. The same manipulative programs now used in deducing the phylogenetic tree from the data can be turned to the automated retrieval of sequence data.

METHODS FOR RECOGNIZING DISTANTLY RELATED SEQUENCES

At present the sequences of almost 500 proteins more than 5 percent different from one another are known.[3] These proteins can be clustered into 116 superfamilies by using statistical methods to recognize their similarity.[18]

A comparison of sequences is based ultimately upon the sum of scores derived from comparisons of an amino acid in one sequence with one in the other sequence. The contribution for each pair of amino acids is specified in a matrix of amino-acid-pair scores. The simplest such matrix counts 1 for amino acid identities and 0 for nonidentities. A very sensitive matrix, derived from the large body of protein mutation data, has proven to be most satisfactory for detecting distantly related sequences.[19]

The most useful computer algorithm for detecting distant relationships[3,19,20] determines the highest possible score for any alignment (including gaps) of two protein sequences. The score for a pair of real sequences is compared with the distribution of scores obtained by aligning 100 pairs of randomized sequences having the same amino acid composition as the two real sequences. This distribution is close to normal. The difference between the score obtained with the two real sequences and the mean score from the randomized sequences is divided by the standard deviation of the random scores to give the "alignment score" in standard deviation units. Comparisons of real sequences that are unrelated give an essentially normal distribution of scores. Probabilities derived from a normal distribution are associated with the alignment scores. When a pair of sequences gets an alignment score of 4.8 SD units or more, we feel quite confident of a relationship ($P<10^{-6}$), presumably by descent from a common ancestor. Because over 6,600 comparisons between superfamilies are necessary for the organization of the presently known data, the standard is set sufficiently high to eliminate false-positive results. Alignment scores are shown in Table I for representatives of the various families of the globin superfamily. The α and β chains form the hemoglobin molecule of vertebrate red blood cells whereas myoglobin is found in vertebrate muscle. The lamprey and annelid globins are also found within blood cells. The insect globin, on the other hand, is dissolved in the body fluid of larval forms. The plant globin is found in the root nodules of a legume.

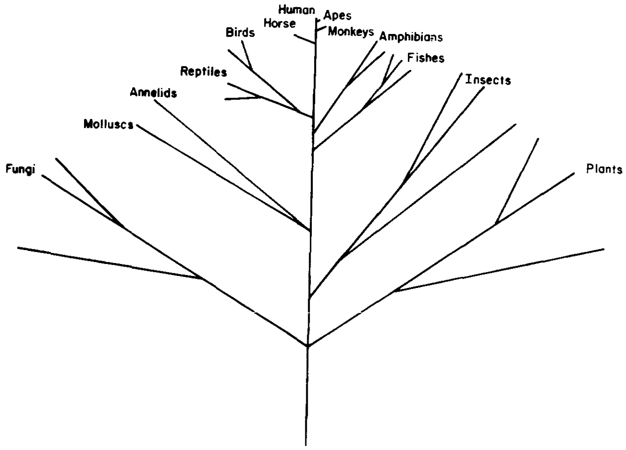


Figure 2—Phylogenetic tree showing the divergences of selected kingdoms, phyla, classes, orders, and families from the human line. The lengths of the branches have been drawn proportional to time without regard to the change in the proteins. The plants, animals, and fungi diverged from each other about 1 billion years ago

EVOLUTIONARY TREE OF A PROTEIN SUPERFAMILY

Each point on the evolutionary tree of a superfamily of related proteins represents a time, a biological species, and a protein structure. There is a point of earliest time on such a tree corresponding to the species ancestral to all of the others represented and within which the ancestral protein sequence for the superfamily was found. Sometimes during evolution a gene

TABLE I—Alignment Scores for Representative Sequences of the Various Globin Families (in SD units)

	α chain	β chain	Myoglobin	Lamprey	Annelid	Insect	Plant
Human Hemoglobin α chain		15.8	8.0	4.9	6.3	2.2	4.7
Human Hemoglobin β chain	15.8		7.6	3.8	6.0	2.9	4.2
Sperm Whale Myoglobin	8.0	7.6		3.6	5.7	3.7	2.7
Lamprey globin	4.9	3.8	3.6		1.2	3.5	1.4
Annelid globin	6.3	6.0	5.7	1.2		3.3	5.3
Insect globin	2.2	2.9	3.7	3.5	3.3		3.1
Plant globin	4.7	4.2	2.7	1.4	5.3	3.1	

is added within an ancestral species through the duplication of a gene already present. Two related proteins, frequently designated by Greek letters such as α and β, may then be found in all descendants. At each point of duplication, the phylogenetic tree of species must be modified to represent the evolutionary tree of the superfamily. The tree of descendant species is duplicated; one subtree represents the α protein and the other, the β protein. There may also be ancestral organisms that have lost genes. The branches for descendant species would then be deleted from the tree. Figure 3 shows the globins to be found in selected vertebrates. The duplication of the tree when the hemoglobin-myoglobin duplication occurred and again when the α-β gene duplication occurred can be seen. The γ chain has been found only in primates and must have been lost in the horse line.

FREQUENCY OF ACCEPTED POINT MUTATIONS

From the number of differences between homologous sequences in biological lines for which divergence times are known from the fossil record, the average rate of change due primarily to point mutations can be derived for each group of proteins. In Table II the average rates of mutation acceptance within selected protein families are shown. Observed changes have been corrected for inferred superimposed mutations. Most values are based on the time interval to 75 million years ago estimated for the mammalian radiation. Although there is more than a 500-fold difference in rates between the slowest and the fastest changing families, the rate of change of proteins within a family seldom varies by more than a factor of 2 or 3, particularly when the proteins fill the same functional niche in different organisms. In the presently available data, the most strongly conserved family is eukaryote histone IV; only two differences in 102 residues are found between the pea and the bovine sequences. The most rapidly changing protein family is amyloid A; its normal function, if any, is unknown, but its abnormal production and deposition is pathological. Even its rate is only one point mutation/100 residues/2 million years. This rate of change is so slow that homologues of all such proteins that were present in the first ancestral vertebrate should still be recognizable in living vertebrates as members of the same protein superfamily. Proteins within a few superfamilies, such as cytochrome c, have changed so slowly that members are recognizable in the whole world of living organisms.

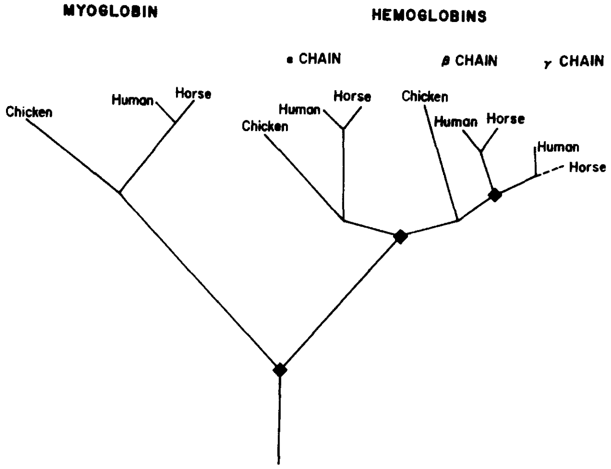


Figure 3—Evolutionary tree of the globin superfamily sequences from vertebrates. The lengths of the branches have been drawn proportional to time without regard to the change in the proteins. Diamonds mark the places where gene duplications have occurred in ancestral forms. The phylogenetic tree of descendant species is repeated at each of these points. The hemoglobin γ chain has been lost in the horse

TABLE II—Average Rates of Mutation Acceptance

Protein Family	PAMs/ 100my*	Detection Range
Amyloid A	48	All vertebrates
Growth hormone	32	
Immunoglobulin C and V regions	32	
Luteinizing hormone β chain	26	
Luteinizing hormone α chain	20	All eukaryotes
Hemoglobin chains	14	
Myoglobin	13	
Animal lysozyme	10	
Thyrotropin β chain	7	All organisms
Trypsinogen	5	
Cytochrome c	2.3	
Glyceraldehyde 3-PO$_4$ dehydrogenase	2.1	
Glucagon	1.1	
Glutamate dehydrogenase	0.90	
Histone IV	0.09	

* *Accepted point mutations per 100 residues per 100 million years*

NOTE: Observed changes have been corrected for superimposed mutations. In most cases estimated rates are based on the divergence of the mammalian orders 75 million years ago. For some proteins the estimated divergence times of other lines were used: amyloid A proteins from man and rhesus monkey, 20 mya; glutamate dehydrogenase from chicken and bovids, 300 mya; cytochrome c and trypsinogen from fish and mammals, 400 mya; glyceraldehyde 3-PO$_4$ dehydrogenase from pig and lobster, 800 mya; histone IV from plants and animals, 1,000 mya

The detection range is inferred from a model of the point mutation process, without insertions and deletions, using the average mutation rate shown. If such a conservative process has always obtained, all of the proteins in the last group could be detected in all living organisms in which they occur. Sequences of trypsinogen, cytochrome c, glyceraldehyde 3-PO$_4$ dehydrogenase, and glutamate dehydrogenase from prokaryotes and eukaryotes are detectably related. Homologues of thyrotropin β chain, glucagon, and histone IV have not been reported from prokaryotes.

For all sequences so far reported, related forms are found in other species. It is expected that related forms of virtually all human sequences would be recognizable in other vertebrates.

FREQUENCY OF INSERTION-DELETION EVENTS

The successful insertion or deletion of one or a few amino acids in a sequence is much less frequent than the acceptance of a point mutation. An average value of one insertion-deletion event per 42 point mutations has been derived from the evolutionary trees of 14 different groups of proteins.[21] It is possible to estimate this frequency with more precision in protein superfamilies, such as eukaryote cytochrome c, for which there is extensive sequence data.

ESTIMATION OF ANCESTRAL SEQUENCES AT THE NODES OF A TREE

Good computer algorithms are available for determining the ancestral sequences at the nodes of an evolutionary tree of protein sequences.[13,16,22] For each possible set of nodal sequences, the tree size is determined by counting the total number of differences between each pair of nodal and terminal sequences that are connected. The one (or set of alternative) nodal sequence(s) that corresponds to the minimum tree size is generated. Sequences along a branch can be predicted as an average of the sequences inferred or known for the two ends of the branch. For the cytochrome c sequence at the divergence of plant, animal, and fungal lines, 86 percent of the positions can be uniquely predicted (with >50% probability of being correct); and for the hemoglobin α chain at the divergence of the chicken and mammalian lines 84 percent of the sequence can be inferred.[1]

ESTIMATION OF PROTEIN SEQUENCES IN ORGANISMS NOT STUDIED

From the phylogenetic tree of living organisms, modified for the gene duplications within the protein superfamily under consideration, a subtree can be abstracted containing the branches for which sequences are known and the branch for which a sequence is sought. The elapsed time corresponding to this latter branch is also known and hence the number of point mutations and gap events expected to occur on it can be calculated from the rate parameters. The inferred ancestral sequence to be thus modified can also be estimated as described above.

Consider as an example a tree involving five species and the first six residues of their hemoglobin α chains, as shown in Figure 4. It is required to estimate the corresponding sequence in the cat. The minimal size tree has the ancestral sequence VLSAAD at all three nodes and also along the branch to which the cat line is connected. There has been one change on the branch to human and two on the branch to the mouse. These three mutations have occurred in the 415 million years of evolutionary time represented by all of the branches on this tree. If the sequences have been equally mutable during this time, then one would predict, using the binomial distribution, that, following the 75 million years represented by the dashed branch to the cat, there is a probability of 0.58 that the cat sequence has remained VLSAAD, a probability of 0.32 that there has been one change, and a probability of 0.10 that there have been two or more changes in the sequence.

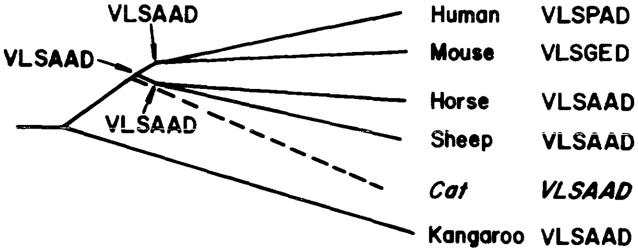


Figure 4—Prediction of the first six residues of the hemoglobin α chain in cat from sequences observed in a marsupial and in four other mammals. The ancestral sequence at all of the nodes is VLSAAD for the most parsimonious tree. The cat sequence is also possibly VLSAAD, although there is a probability of 0.55 that at least one residue has changed

In this procedure there are two major contributions to the error in the estimates, the error in the estimate of the mutation rate and the prediction error of the model. In this example, the standard deviation of the count is approximately $\sqrt{n}$, corresponding to a probability of 0.58±0.18 that no change will be found in the cat sequence. The precision of the model, which is derived from proteins in all superfamilies, is difficult to estimate from the data now available, but it seems close to the expectation for independent events. Additional parameters can be included as more data accumulates.

CONCLUSION

The conceptual tools and the procedures necessary for a prediction of all of the 10^{10} or 10^{11} protein sequences in living species are already highly developed. One can readily visualize the separate parts operating as an integrated interactive computerized system that could predict sequences for specified organisms with an estimated precision from the collection of known sequences.

REFERENCES

1. Dayhoff, M. O., editor, *Atlas of Protein Sequence and Structure 1972*, Vol. 5, National Biomedical Research Foundation, Washington, D.C., 1972.
2. Dayhoff, M. O., editor, *Atlas of Protein Sequence and Structure*, Vol. 5, Suppl. 1, National Biomedical Research Foundation, Washington, D.C., 1973.
3. Dayhoff, M. O., editor, *Atlas of Protein Sequence and Structure*, Vol. 5, Suppl. 2, National Biomedical Research Foundation, Washington, D.C., 1976, in press.
4. Dayhoff, M. O., L. T. Hunt and W. C. Barker, *Protein Sequence Data Tape 76*, National Biomedical Research Foundation, Washington, D.C., 1976.
5. Dayhoff, M. O., L. T. Hunt, W. C. Barker and B. C. Orcutt, *Protein Segment Dictionary 76*, National Biomedical Research Foundation, Washington, D.C., 1976.
6. Dayhoff, M. O., P. J. McLaughlin, W. C. Barker and L. T. Hunt, *Naturwissenschaften* 62, pp. 154-161, 1975.
7. Dayhoff, M. O., *Fed. Proc.* 1976, in press.
8. Dayhoff, M. O., "Evolution of proteins," in *Exobiology*, edited by C. Ponnamperuma, pp. 266-300, North-Holland, Amsterdam, 1972.
9. Margulis, L., *Origin of Eukaryotic Cells*, Yale Univ. Press, New Haven and London, 1970.
10. Margulis, L., *Biosystems* 7, pp. 266-292, 1975.
11. Schwartz, R. M., W. C. Barker and M. O. Dayhoff, in *Second College Park Colloq. on Chemical Evolution*, 1975, p. 40, Univ. Maryland, College Park.
12. McLaughlin, P. J. and M. O. Dayhoff, *J. Mol. Evol.* 2, pp. 99-116, 1973.
13. Dayhoff, M. O., C. M. Park and P. J. McLaughlin, in *Atlas of Protein Sequence and Structure 1972*, Vol. 5, edited by M. O. Dayhoff, pp. 7-16, National Biomedical Research Foundation, Washington, D.C., 1972.
14. Dayhoff, M. O., L. T. Hunt, P. J. McLaughlin and D. D. Jones, in *Atlas of Protein Sequence and Structure 1972*, Vol. 5, edited by M. O. Dayhoff, pp. 17-30, National Biomedical Research Foundation, Washington, D.C., 1972.
15. Langley, C. H. and W. M. Fitch, *J. Mol. Evol.* 3, pp. 161-177, 1974.
16. Moore, G. W., J. Barnabas and M. Goodman, *J. Mheor. Biol.* 38, pp. 459-486, 1973.
17. McLaughlin, P. J., L. T. Hunt and M. O. Dayhoff, *J. Human Evol.* 1, pp. 565-578, 1972.
18. Dayhoff, M. O., W. C. Barker and L. T. Hunt, in *Atlas of Protein Sequence and Structure*, Vol. 5, Suppl. 2, edited by M. O. Dayhoff, National Biomedical Research Foundation, Washington, D.C., 1976, in press.
19. Barker, W. C. and M. O. Dayhoff, in *Atlas of Protein Sequence and Structure 1972*, Vol. 5, edited by M. O. Dayhoff, pp. 101-110, National Biomedical Research Foundation, Washington, D.C., 1972.
20. Needleman, S. B. and C. D. Wunsch, *J. Mol. Biol.* 48, pp. 443-453, 1970.
21. Dayhoff, M. O. and W. C. Barker, in *Atlas of Protein Sequence and Structure 1972*, Vol. 5, edited by M. O. Dayhoff, pp. 41-45, National Biomedical Research Foundation, Washington, D.C., 1972.
22. Fitch, W. M., *Syst. Zool.* 20, pp. 406-416, 1971.

From text to structured information—Automatic processing of medical reports*

by LYNETTE HIRSCHMAN, RALPH GRISHMAN and NAOMI SAGER
New York University
New York, New York

ABSTRACT

This paper describes the analysis and processing programs for a set of natural language texts in a medical area (x-ray reports on patients with breast cancer). The programs convert the information in the text into a tabular form suitable for further automatic information processing (e.g., editing of records, question answering on the data collected, or statistical summaries of the data). To set up a tabular form appropriate for the data, we first perform a manual linguistic analysis on a sample of the texts. From this we obtain the word classes and the form of the table (called an information format) for this type of material. We then apply the series of processing programs to the sentences of the texts. Each sentence is parsed with the Linguistic String Parser English grammar in order to obtain its grammatical structure; certain standard English transformations are then applied to regularize the grammatical form of the sentence; and finally a set of "formatting transformations" map the words of the sentence into the slots of the format or table, in such a way that the sentence is reconstructible (up to paraphrase) from its representation in the table. The results of applying these programs to a corpus are described. This procedure enables us to convert a natural language corpus into a structured data base.

INTRODUCTION

An essential part of the effective management of scientific and technical information is the efficient retrieval of information from a large body of text. One example of this is the retrieval of documents from a large collection of scientific articles, in response to a user's request. Another example of the same problem is the extraction of data from medical reports for statistical purposes, or for fact retrieval.

The key to efficient retrieval lies in the appropriate structuring of the information. For document retrieval, this may involve the extraction of key terms for each document. For medical records, it may involve transferring the most essential information into separate tables. These tasks pose a considerable burden on the preparer of the document. In addition, each such structuring will be appropriate only for the retrieval of certain types of information from the data base.

What is required therefore is a procedure for the automatic structuring of the natural language material itself, in such a way that all the information is preserved. The Linguistic String Project of New York University has been engaged in a long-term effort to develop techniques for processing textual information. These techniques are based on distributional analysis and computerized parsing of English texts. We intend in this paper to give an overview of our approach and to describe briefly our latest experiments.

OVERVIEW

Since we are dealing with textual data, structuring the information means, first of all, structuring the sentences. The question then is: what sort of structure should be assigned to the sentences? One alternative is some kind of surface parse tree. PROTOSYNTHEX I,[1] one of the earliest systems for information retrieval from natural language texts, attempted to use dependency analysis to match requests for information with sentences in the data base. However, surface analysis alone is inadequate for such information processing; one limitation is that it does not take into account possible differences between data and request due to grammatical paraphrase. For ex-

* This investigation was supported in part by research grant 1-RO1-LM-02616 from the National Library of Medicine, National Institutes of Health, DHEW, in part by Public Health Service Research Grant No. CA-11531 from the National Cancer Institute, and in part by research grant SIS75-22945 from the National Science Foundation, Office of Science Information Service. Development of the program for transformational analysis was supported in part under Contract No. NOOO14-67A-0467-0032 with the Office of Naval Research.

ample it would fail to match a request stated as an active sentence with an otherwise identical sentence in the data base which was in the passive voice.

It has long been recognized[2] that the effects of such paraphrastic variation can be overcome by performing some type of transformational analysis on the sentence. Transformational decomposition, following the theory of Harris, or deep structures, following the theory of Chomsky, can be used to reduce grammatical paraphrases to a standard form. A Linguistic String Project study in 1970 showed that Harrisian transformational decompositions could be useful in matching technical articles with information requests.[3] Such techniques can be used to structure a variety of texts; however, the resultant structures provide only general grammatical relations (subject, object), which are not directly related to the semantic or informational classes in a specific scientific subfield. In other words, the categories of English grammar are too general for information structuring.

It is possible to write a grammar specific to the use of language in a particular subfield of science, employing the same methods used to write descriptive grammars of whole languages. The resulting sublanguage grammar yields structures suitable for information processing: the word classes of this grammar are the word classes of semantic interest in the subfield; the overall arrangement of classes provides a format for the information content of subfield text sentences. For example, the grammatical structure of medical reports includes categories for patient, type of test, body organ tested, date of test, etc. Such an organization can greatly facilitate information retrieval or statistical manipulation of the data. On the other hand, each scientific field and type of text has its own structure. This means that a detailed linguistic analysis is required every time a new class of text is to be handled.

In this paper we describe an experiment in the computer formatting of material from medical records. Our previous papers have described the method of sublanguage analysis and information formatting for more complex textual material,[4,5] as well as the battery of programs which have been developed for text processing.[6,7] Here we focus on the problem of mapping text sentences into information formats. In the sections which follow, we will describe how the format for a particular type of medical narrative was derived, and how sentences are automatically transformed into structured information, as specified by the format. We will also indicate how the process of deriving formats may be automated or partially automated, and how the structured information of the formatted sentences can be used.

THE TEXTS

For our initial experiment in the computer formatting of texts, we chose to work on medical records. A set of follow-up reports on patients with cancer was provided to us in machine readable form, as part of a collaborative research project with Dr. I. D. J. Bross of Roswell Park Memorial Institute. The reports included laboratory tests, pathology reports, radiology reports, records of treatment, and discussion of medical problems. A linguistic analysis of some of these reports was done at Roswell Park.[8] We chose to process one particular type of report, identified as "Findings R(adiology)." This material was selected because it contained both full sentences and sentence fragments, a combination typical of the compressed note-taking style of much medical narrative (e.g., *x-rays not taken,* or *nothing to indicate metastasis*). The limited vocabulary of Findings R and the frequent paraphrasing of the various types of medical information made it possible to define valid word classes and formats on a limited corpus.

The corpus consisted of 159 Findings R reports on 11 patients, containing a total of 188 sentential units.* Due to frequent repetition of certain formulaic expressions, such as *x-rays negative,* only about half of the sentential units (86/188) were distinct, ignoring differences in date.

CREATION OF SUB-LANGUAGE FORMATS

To convert the medical information contained in a sentence into tabular form, we create a table (or format) with slots for each class of relevant information. The definition of a set of formats for a particular sub-field is done in two steps: first we perform a distributional analysis on the parsed sentences to obtain the sub-language word-classes; we then use the distribution of the word-classes to define the formats.

Distributional analysis involves classifying together words which occur in the same syntactically defined environments; for example verbs which occur with the same subjects and objects would form a word class. We begin building each class by finding a few words which occur frequently and share a number of environments. These words form the "core" of the new class. We then enumerate the environments in which these core words occur, and look for other words which share some of the same environments. If these other words occur primarily in the same environments as the core words, we add them to the class. This process can be illustrated with the NTEST (Noun TEST) class. The words *x-ray* and *film* share many environments, and are thus selected as the core of a new class. The characteristic environments in which they occur are:

* A sentential unit is a word sequence ending in a period; a sentential unit may contain more than one sentence or sentence fragment: *chest x-ray unchanged, nothing to indicate metastatic disease.*

$$(1)\quad [\text{chest}]\begin{Bmatrix}\text{x-ray(s)}\\ \text{film(s)}\end{Bmatrix}[\text{RN}]\begin{bmatrix}\text{show}\\ \text{--}\end{bmatrix}\; [\text{LN}]\begin{Bmatrix}\text{change(s)}\\ \text{metastasis}\\ \text{metastases}\end{Bmatrix}[\text{RN}]$$

$$(2)\quad [\text{chest}]\begin{Bmatrix}\text{x-ray(s)}\\ \text{film(s)}\end{Bmatrix}[\text{RN}]\begin{bmatrix}\text{be}\\ \text{--}\end{bmatrix}\; \text{negative } [\text{RN}]$$

Here braces enclose alternative elements and brackets enclose optional elements; LN and RN designate left and right adjuncts (modifiers) of the noun. Note that the dash is treated as a word of the sentence. Looking for other words which appear in these environments, we find:

(a) *Mammograms* no change. . . .
(b) Metastatic *series* showed extensive osteolytic metastases. . . .
(c) Metastatic bone *survey* -- negative.
(d) Flat *plate* -- mild degenerative changes. . . .
(e) Flat *plate* of abdomen -- shows lumbar spine to be riddled with multiple metastatic areas.

Since the environments of *mammograms, series, survey,* and *plate* match either environment (1) or (2) of the core words, they are added to the NTEST class.

We have implemented this approach to word classification in a computer program, although using a somewhat different procedure than that described above.[9] The program has been applied to the Findings R data and to other texts. Both the manual and computerized methods successfully classify all of the frequent words and some of the infrequent words.

To capture more of the infrequent words, we use a second-order distribution analysis procedure. In the characteristic environments of each class, we replace each word which has already been classified, by the name of its class. Consider the two environments of *x-ray* and *film* given above. At this point *chest* has been assigned to the NBODY class; *x-ray* and *film* to the NTEST class; *show* to the VSHOW class; *be* to the VBE class; *change* to the NCHANGE class; *metastasis* (*-ses*) to the NCONDITION class; and *negative* to the NONPATHADJ (non-pathological adjective) class. Replacing each word in the environments listed above by its class name, we get:

$$(3)\quad [\text{NBODY}]\ \text{NTEST}\ [\text{RN}]\begin{bmatrix}\text{VSHOW}\\ \text{--}\end{bmatrix}\; [\text{LN}]\begin{Bmatrix}\text{NCONDITION}\\ \text{NCHANGE}\end{Bmatrix}[\text{RN}]$$

$$(4)\quad [\text{NBODY}]\ \text{NTEST}\ [\text{RN}]\begin{bmatrix}\text{VBE}\\ \text{--}\end{bmatrix}\; \text{NONPATHADJ}\ \ [\text{RN}]$$

In similar fashion, we take the environments of each unclassified word and replace the words by class names where possible. For instance, there are two occurrences of *scan*:

(f) The liver scan was normal.
(g) Brain scan shows midline lesion.

Replacing words by class names, we obtain:

(h) NBODY scan VBE NONPATHADJ
(i) NBODY scan VSHOW LN CONDITION

Since these two sentences match the environment for NTEST words, we add *scan* to the NTEST class.

There are some words which occur so infrequently (once or twice in the corpus) that we cannot rely on distributional analysis to classify them. However these words must be assigned to a sublanguage class if the sentences in which they occur are to be correctly formatted. (Words are assigned to format slots on the basis of membership in a word class.) In these cases we either extend the criteria of a sub-class in reasonable ways, or if all else fails we use our knowledge of the meaning of a word to fit it into a subclass. On this basis we add to the NTEST class the words *auscultation, percussion, urinalysis,* and *view,* each of which occurred only once in the corpus.

Once we have defined the sublanguage word classes, we can use the word classes to define the sublanguage formats. A format is constructed so that:

1. equivalent pieces of information in different sentences will map into the same format slots;
2. each informationally significant word in a sentence is mapped into a separate slot of the format;
3. in each sentence, certain slots of the format may be empty, if the sentence does not contain that particular type of information;
4. not every word in a sentence will receive its own format slot: certain modifiers (e.g., *the*) are simply left as adjuncts on their head noun, if they contain no sublanguage information, or if they never occur independently of a particular word class;
5. *all* the words of the sentence are mapped into the format, preserving their original order of occurrence, with the exception of certain allowable paraphrastic permutations.

Once the sentences are formatted, we know exactly where (what slot or slots) to check, in order to find any particular type of information, in any sentence. However the formatted sentence will resemble the original unformatted sentence very closely, since no words are lost, and word order is preserved up to paraphrase. It is surprising that the sublanguage sentences are so highly structured that an information format can be constructed in this way, but it is just this structure that makes it feasible to do natural language processing on these texts.

We begin to build the format by taking a sentence of the corpus:

(a) Chest x-ray 12-6 shows no evidence of metastasis.

We replace the words by their sublanguage classes:

$$\text{(b)}\ \underset{\text{subj}}{\begin{bmatrix}\text{NBODY} & \text{NTEST} & \text{DATE}\\ \text{chest} & \text{x-ray} & \text{12-6}\end{bmatrix}}\ \underset{\text{verb}}{\begin{bmatrix}\text{VSHOW}\\ \text{shows}\end{bmatrix}}$$

$$\underset{\text{obj}}{\begin{bmatrix}\text{NEGATIVE} & \text{NSHOW} & \text{P} & \text{NCONDITION}\\ \text{no} & \text{evidence} & \text{of} & \text{metastasis}\end{bmatrix}}$$

Each significant word gets its own slot. Of these words, only P (preposition) has no significance beyond its role as syntactic marker; it is therefore included as an adjunct of NCONDITION. We can now write our first tentative format. The format slots are given names related to the type of information they will contain. The gross syntactic structure of the parsed sentence provides some additional groupings of the format slots into TEST (subject) and FINDING (predicate). In Table I words in () are adjoined to the main word in the slot.

Next we take another sentence and again replace the words by their word classes:

$$\text{(c)}\ \begin{bmatrix}\text{DATE} & \text{NTEST} & \text{P} & \text{ADJSPINE} & \text{NBODY}\\ \text{10-26} & \text{film} & \text{of} & \text{lumbar} & \text{spine}\\ \text{subj} & & & & \end{bmatrix}$$

$$\begin{matrix}\text{--}\\ \\ \text{--}\end{matrix}\begin{bmatrix}\text{NEGATIVE} & \text{NCHANGE}\\ \text{no} & \text{change}\\ \text{obj} & \end{bmatrix}$$

The subject of sentence (c) contains the word classes DATE, NTEST, NBODY, but in a different order than sentence (b). However there are paraphrastic transformational relations that allow the date (a time expression) to be on either side of the subject; and a paraphrastic transformational relation between the two noun phrases:

$$\begin{array}{ccccc} N_1 & N_2 & \leftrightarrow & N_2 \text{ of } & N_1\\ \text{chest} & \text{film} & & \text{film of} & \text{chest}\end{array}$$

Since the subjects of sentences (b) and (c) contain the same kinds of information, this information must be mapped into the same format slots in both cases. Changing the word order of sentence (c) for formatting is permissible, since it only involves paraphrastic permutations. Therefore *10-26*, *film*, and *spine* map into the format slots TESTDATE, TESTN, and TESTLOC as set up in format #1. *Lumbar* is not assigned a format slot of its own, but is left as an adjunct on *spine*, because ADJSPINE adjectives occur only on *spine* in this corpus; that is, they have no independent status and do not get a separate column of the format.

Next we must decide what to do with the symbol "--" which appears between subject and object in sentence (c). Should it be assigned to a new format slot, or can it be mapped into the VERB category? If we examine its distribution, we find that it has the distribution of VSHOW in certain cases, and of VBE in others, e.g., *Chest x-ray -- no evidence* and *Chest x-ray -- negative*. It is therefore appropriate to map it into the VERB slot. Finally, we must decide where to put NCHANGE in the format. Its distribution differs from NSHOW and NCONDITION; in particular it can occur in the same sentence with words from these two classes:

(d) No evidence of recurrence
NSHOW NCHANGE

(e) No callus formation
NCONDITION NCHANGE

Clearly the class NCHANGE is not in complementary distribution with either NSHOW or NCONDITION. We must create a new slot in the format between INDICATION and MED-FINDING to house it. Our revised format #2 is shown in Table II with formatted sentences (b)-(e):

In this manner we build up the format on the basis of a limited number of sentences. The adequacy of the format created can be tested by using it in the formatting of a different set of texts. The x-ray format made up from part of the Findings R data has been tested both against other Findings R data and against a different set of x-ray data from patients with sickle cell disease. In both cases it was found adequate to format the radiology material.

TABLE I

FORMAT #1

TEST			FINDING			
TESTLOC	TESTN	TESTDATE	VERB	NEG	INDICATION	MED-FINDING
NBODYPT	NTEST	DATE		NEGATIVE	NSHOW	NCONDITION
chest	x-ray	12-6	shows	no	evidence	(of)metastasis

TABLE II

FORMAT #2

	TEST			FINDING				
	TESTLOC	TESTN	TESTDATE	VERB	NEG	INDICATION	CHANGE	MED-FINDING
	NBODYPT	NTEST	DATE	VSHOW --	NEGATIVE	NSHOW	NCHANGE	NCONDITION
b)	chest	x-ray	12-6	shows	no	evidence		(of)metastasis
c)	(of) (lumbar) spine	film	10-26	--	no		change	
d)					no	evidence	(of) recurrence	
e)					no		formation	callus

Not all the entries in Findings R report the results of a test. There are a few sentences that refer directly to the patient:

(g) Patient given penicillin for 9 days.

(h) Patient to return in one month for repeat x-ray.

Clearly these sentences require a different format from the one being developed above. Since there are so few sentences of this type, a much larger corpus would be required to define a format for sentences (g) and (h), but as these sentences illustrate, even in a restricted subfield of a medical report, several formats may be needed to represent the different types of information encountered.

FORMATTING THE TEXT

Once the format is defined, the sentences must be mapped into the format. As before, it is important to have a procedure which can be generalized to texts in other subfields. Our procedure is built around the Linguistic String Parser, a powerful system for language analysis which provides the mechanism for parsing sentences with a context-free grammar augmented by restrictions;[7] it also provides the machinery for performing transformations on parsed sentences,[10] and a higher level language (the Restriction Language) for writing restrictions and transformations.[11]

Sentence formatting is done in three stages:

1. determination of sentence structure by linguistic string analysis;
2. regularization of certain sentence structures by use of general English transformations;
3. mapping of transformed parsed sentences into format slots, using specialized "formatting transformations."

We will briefly consider each stage in turn.

Linguistic string analysis

Linguistic string analysis provides a structural description of the sentence in terms of a specified set of linguistic strings. The assignment of a word to a format slot depends on its role in the sentence structure, as well as on its word class, so that a determination of sentence structure is a prerequisite to formatting. For example syntactic analysis resolves part-of-speech ambiguities, so that the word *left* is identified as a verb in

(a) The patient left the hospital.

but as an adjective in

(b) X-ray of the left lung showed metastasis.

The LSP string grammar was originally designed to handle only complete English sentences; it provides a broad coverage of English syntactic constructions and together with its associated word dictionary, has been used to analyze English scientific texts. In order to process the note-style and incomplete sentences of the medical reports, we made four changes in the grammar.

First, the grammar was expanded to handle the sentence fragments by adding a small number of new productions to the context-free component. Five types of fragments were allowed.

1. A sentence with subject and object but either without verb or with a dash (--) in place of a verb: *Chest x-ray -- no change., 10-6 x-ray negative.*
2. An adjective with its adjuncts: *Negative for metastatic disease.*
3. A noun with its adjuncts: *No evidence of change.*
4. A passive sentence without subject or *be*: *Not done on previous exam.*
5. A sentence preceded by a noun phrase. *Chest x-ray 4-6-71 chest film shows no evidence of fluid.*

Second, one restriction in the grammar was removed in order to accommodate the note-taking style of the text: this was the count noun restriction, which requires that a singular count noun have an article or some other appropriate form of modifier before the noun. For example, the Findings R text contains sentences like *X-ray shows lesion,* whereas in normal English both *x-ray* and *lesion* must be preceded by an article: *The x-ray shows a lesion.*

Third, certain constructions that were unlikely to occur in this type of text were eliminated from the grammar, for example, the question constructions. This pruning of the grammar speeded up the sentence analysis considerably.

These first three changes were designed to accommodate texts in a note-taking style and would be applicable to any subject area. A fourth change, needed to handle certain types of ambiguity, required the use of word classes and selectional restrictions specific to the sublanguage grammar of radiology reports.

One such type of ambiguity is a predictable structural ambiguity, which must be resolved in order to format the sentences correctly. This type of ambiguity can arise from modifiers on conjoined material. For instance, the sentence

(c) X-rays of lumbar spine and chest showed lesions.

may be analyzed as any one of the following:

(d) X-rays of lumbar spine showed lesions and chest showed lesions.

(e) X-rays of lumbar spine showed lesions and x-rays of lumbar chest showed lesions.

(f) X-rays of lumbar spine showed lesions and x-rays of chest showed lesions.

Such ambiguity is inherent in the syntactic construction, and has nothing to do with the particular words involved. Only sublanguage selectional restrictions can resolve it. In this example, *lumbar* is an ADJ-

SPINE which modifies only the noun *spine*, eliminating reading (e). Since *spine* and *chest* are both NBODY, it is more likely that they are conjoined than words of different classes (e.g., *x-ray*, an NTEST and *spine*). This eliminates reading (d) leaving the correct reading (f).

Another type of ambiguity arises from an "over-rich" lexicon--a lexicon for all of English, containing possible uses of words that would never occur in this sublanguage. The sentence

(g) No report of x-rays being taken.

received a parse in which *being* was taken to be a noun (as in *human being*) which was the object of a missing verb *be* or *show* derived from a sentence:

(h) No report of x-rays shows a being which has been taken.

There were several ways to deal with this kind of ambiguity. We could have used selectional restrictions on the subject and object of *taken*, or we could have placed tighter restrictions on the construction with an omitted verb. However we chose what seemed the simplest and most direct approach for such cases: we created a special x-ray dictionary, by editing the general English dictionary to remove word classifications (e.g., *being* as a noun) which would never occur in the Findings R text.

English transformations

In the Linguistic String Parser, the transformations are applied to the output of the string analysis. The function of the transformations is to regularize the parse trees, reducing the variety and complexity of structures present. For example, the sentences

(i) X-rays of chest and pelvis negative.
(j) X-rays of chest negative and x-rays of pelvis negative.

contain the same information. By transforming the parse tree for the first sentence into the tree for the second sentence we produce a more regular set of structures in which only full sentences (or sentence fragments) are conjoined. We also transform relative clauses into complete sentences; for example, we would convert

(k) X-rays showed a lesion which may be metastatic.

to

(l) X-rays showed a lesion such that the lesion may be metastatic.

The gain achieved in performing this transformation is that the complete sentences derived from relative clauses can then be formatted in the same way as any other sentence; no special process for formatting relative clauses is required.

These transformations are written for all of English; they do not make use of any information specific to the Findings R sublanguage. There are a large number of English transformations, but only a very few have been used in this application. This is because transformations expand compressed material into a more regular form by filling in certain pieces of redundant information, or information retrievable from context (like the verbs *be* or *show*). If a particular type of information is always omitted in a certain class of texts, no regularization is achieved by trying to fill in this missing information. For example, the word *x-ray* can be used both as a verb and as a noun, so we could have an English transformation to convert sentences with the noun to sentences with the verb; in Findings R, however, *x-ray* is used only as a noun, and no regularity would be gained. Moreover, the verb requires a subject--the taker of the x-ray--which is never present in this text. As a result, the only two English transformations used are the conjunction expansion and relative clause expansion described above. However, in more syntactically complex material or less abbreviated material, there might be a real benefit from a greater regularization of the syntax (via transformations) before attempting to format it.

Formatting transformations

The formatting transformations transfer the words from the parsed sentences to the appropriate slots in the format. They use the same transformational mechanisms built into the Linguistic String Parser to handle the English transformations. Because these mechanisms are set up to map trees into trees, the format is first created as a tree; after it has been built, it can be written out in the tabular form shown in Tables I-III. Formatting transformations move the words from the original ASSERTION or FRAGMENT node in the parse tree into the format slots. As a result, at the end of the formatting process, the FORMAT has the words of the sentence in it, while the original ASSERTION or FRAGMENT node is empty. This provides a check on the completeness of the formatting process.

Three kinds of transformations can be distinguished. The first type of transformation sets up the format slots under the node FORMAT. For sentences which contain a verb or adjective connecting two findings or pieces of data (e.g., *related to, compatible with, typical of*), the format is augmented with a CONNECTIVE slot and an additional set of FORMAT slots. It is necessary to add this new FORMAT to provide an empty set of slots for the second finding. Relative clauses are treated similarly: a CONNECTIVE slot is added, with a relative clause marker placed in the CONN slot under CONNECTIVE; and the assertion contained in the expanded relative clause is mapped into the new set of format slots.

Once the format slots have been set up, the remaining transformations each map words of one class into the appropriate format slot. These transformations fall into two groups. One type requires little if any syntactic or co-occurrence information; it simply searches the parse of the sentence for a word having both a particular syntactic category and a certain sublanguage word class, and then maps the word into the format slot associated with that word class. For example, the T-NTEST transformation looks for a NOUN of class NTEST. When it finds such a word in the sentence being formatted, it moves the word, together with its adjuncts, into the appropriate format slot (TESTN).

A different type of formatting transformation is required for a word that can go into one of several slots depending on what it modifies (e.g., negative and indefinite words). This class of transformations relies heavily on the availability of syntactic information from the parse. For example, *not* can go in any one of three slots, depending on what kind of verb it negates. Therefore the T-NOT transformation must apply before any of the verb transformations have moved the verb into its format slot: co-occurrence relations must be checked in the parse tree, where the syntactic relations are still explicit. Once the verb has been moved into the format, the syntactic relations have been translated into informational relationships and are no longer explicitly expressed. When the T-NOT transformation finds a *not*, it checks the main verb occurring in the same string with *not*. If the verb is VSHOW (e.g., *X-rays do not show metastasis.*), the *not* goes into NEG in FINDING. If the verb is VDONE (as in *not done*) the *not* goes into the NO-TEST slot, under TEST, because the class VDONE occurs only with NTEST nouns; if *not* occurs with VDONE, it necessarily negates the existence of a test, even if no NTEST word occurs in the sentence. Finally if the *not* negates a word that connects two findings (e.g., *is not related to, is not compatible with*) it will go into the NEG-CONN slot under CONNECTIVE.

The set of formatting transformations can be viewed as a set of special sublanguage transformations which reduce various sublanguage paraphrases to a standard representation in the format. For example, to find out if a test was performed, we need only inspect the NO-TEST column of the format. If it is empty, a test has been performed and we can find the type of test by looking in the NTEST slot. Or if we want to know when the first abnormality is seen in a patient, we look for the first sentence where both (1) FINDING is not empty and (2) the columns NEG and STATUS in FINDING are both empty. This is because all the "normal" findings are expressed either by NON-PATHADJ (*negative, normal*) formatted in the column STATUS or by expressions like *no change, no metastasis, no evidence of metastasis.* If one of the slots in FINDING has an entry other than NEG or STATUS, then it must be an INDICATION, a CHANGE, or a MEDical-FINDING (PART-OF-BODY words will not occur by themselves in the FINDING slot). The format thus standardizes the representation of the important medical information in the sentence, so that this information can be further processed.

RESULTS

To each sentence of our corpus we applied the formatting program, which parsed the sentence, performed certain English transformations on it, and then mapped this structure into the format. This program successfully formatted 176 of the original 188 sentences (94 percent). Table III presents the full format and several examples of formatted sentences.

The full format contains sets of slots for OBSERVE (for doctor+verb: *radiologist noted*), TEST and FINDING. For those sentences that require more than one set of format slots to accommodate their information (e.g., sentences 4 and 5 in Table III), additional sets of format slots are added, each linked to the preceding format by a CONNECTIVE:

FORMAT			CONNECTIVE	FORMAT		
DATA				DATA		
OBSERVE	TEST	FINDING		OBSERVE	TEST	FINDING

In Table III, each row represents a set of format slots; a sentence that requires three sets of format slots (e.g., sentence 5) will therefore occupy three lines of the table.

CONCLUSION

The formatting procedure enables us to convert a natural language corpus into a structured data base. Given a set of x-ray reports in machine readable form, the formatting program maps the input sentences one by one into the tabular format structure. This data base can be used in a variety of ways; we are currently at work on a program to extract various medical statistics from the data base (e.g. number of patients with recurrence of metastasis; time from operation to time of first suspected recurrence of metastasis; location of new metastasis, etc.). It should also be possible to use the data base with a natural language front end to process questions and answer them with information from the data base.

The formatting program is able to convert the natural language material into structured information partly because the material chosen for processing is itself highly structured; however, the formatting relies heavily on a linguistic analysis of each sentence, in order to handle such informationally complex structures as relative clauses, negation, and conjunction.

TABLE III

THE COMPLETED FORMAT: OBSERVE and TEST columns, with examples of formatted sentences.

Labels of the format slots:

NOTE: in the formatted sentences, adjuncts are placed in (); left adjuncts appear above the main word, right adjuncts below it.

Findings R sentences, formatted in the columns to the right:

	FORMAT								
	DATA								
	OBSERVE		TEST						
	MD	REPORT	NO-TEST	TEST-VIEW	TESTN	TESTLOC	VERB-DONE	OCCASION	TESTDATE
1. Chest film 6-5 shows enlargement of lesion on right hilum since film of 4-17					film	chest			6-5
2. X-rays taken 3-22-65 reveal no evidence of metastatic disease.					x-rays		taken		3-22-65
3. None this examination			none					(this) examination	
4. Nothing definite is seen that indicates tumor.		(is) seen							
5. The heart, lungs, and bony structures are intact.									

Without a stage of syntactic processing, it would not be possible to determine the scope of negation, the appropriate expansion for conjoined elements, or the referend of the relative pronoun. Because we can process these linguistic structures we can go beyond document retrieval; we are now able to "get inside" the text, to process the actual content.

Our formatting experiment was conducted on a rather simple set of reports which had little paragraph structure and contained specific limited kinds of information. A text that contained several different types of information (requiring several different formats), or had a more complex paragraph structure, with a corresponding increase in intersentential reference, would pose somewhat greater difficulties than the type of material discussed here. Nonetheless there are many instances of natural language material that is both restricted and highly structured (different types of medical reports; weather reports; program specifications in natural language), where this type of procedure would be successful in structuring the information.

Although the specific program described here will process only x-ray reports, the techniques that have been used to obtain the program are general. The string grammar parses English sentences; a few changes enabled it to handle fragmentary note style. The procedure for defining word classes (and selectional restrictions) is based on distributional analysis and can be applied to any language or sublanguage. Part of the procedure for obtaining word classes has been automated in the clustering program;[9] one of our next projects is to complete the automation by adding a program that will convert the parsed sentence into co-occurrence patterns suitable for clustering. The definition of the format was a general technique,

TABLE III—Continued

THE COMPLETED FORMAT: FINDINGS and CONNECTIVE columns.

FORMAT, continued
DATA, continued

	FINDING																			CONNECTIVE		
	NEG	INDEF-FIND	VERB-ELEMENT		CHANGE-OVER-TIME										STATUS	MED-FINDING	REGION			NEG-CONN	INDEF-CONN	CONN
			BE-SHOW	INDICATION	CHANGE	TIME-PERIOD											POSITION	PART-OF-BODY	STRUCTURE			
						WHEN	OBSERVE		TEST													
							MD	REPORT	TEST-VIEW	TESTN	TEST-LOC	VERB-DONE	OCCASION	TEST-DATE								
1.			shows		enlargement	since				film				(of) 4-17		(of) lesion	on	(right) hilum				
2.	no		reveal	evidence												(of) metastatic disease						
3.																						
4.	nothing (definite)																					rel-clause
	nothing (definite)		indicates													tumor						
5.			are												intact			(the) heart				and
			are												intact			(the) lungs				and
			are												intact			bony	structures			

originally developed on a corpus from pharmacology.[4] An overall strategy for mapping parsed sentences into a sublanguage format has been defined, although the transformations themselves are dependent on the target structure (the format) as well as the type of sentence structures in the input. Because each step of our procedure has been based on general linguistic techniques, it should be possible to apply this procedure to convert natural language texts of any sufficiently structured subfield into a structured data base.

REFERENCES

1. Simmons, R., S. Klein and K. McConlogue, "Indexing and Dependency Logic for Answering English Questions," *American Documentation* 15, p. 196, 1964.
2. Harris, Z. S., "Linguistic Transformations for Information Retrieval," *Proc. Int'l. Conf. on Scientific Information* (1958) 2, p. 158, 1959.
3. Sager, N., J. Touger, Z. S. Harris, J. Hamann, and B. Bookchin, "An Application of Syntactic Analysis to Information Retrieval," *String Program Reports* No. 6, Linguistic String Project, New York University, 1970.
4. Sager, N., "Syntactic Formatting of Scientific Information," *Proceedings of the 1972 Fall Joint Computer Conference*, AFIPS Conference Proceedings, Vol. 41, pp. 791-800, AFIPS Press, Montvale, N.J., 1972.
5. Sager, N., "The Sublanguage Technique in Science Information Processing," *Journal of the American Society for Information Science*, Vol. 26, pp. 10-16, 1975.
6. Sager, N., "Syntactic Analysis of Natural Language," *Advances in Computers*, Vol. 8, pp. 153-188, Academic Press, Inc., New York, 1967.
7. Grishman, R., N. Sager, C. Raze, and B. Bookchin, "The Linguistic String Parser," *Proceedings of the 1973 Computer Conference*, pp. 427-434, AFIPS Press, 1973.
8. Anderson, B., I. D. J. Bross and N. Sager, "Grammatical Compression in Notes and Records: Analysis and Computation," paper delivered at the 13th Annual Meeting of the Association of Computational Linguistics, Boston, Nov. 1, 1975, *American Journal of Computational Linguistics*, Vol. 2, No. 4, 1975.
9. Hirschman, L., R. Grishman and N. Sager, "Grammatically-based Automatic Word Class Formation," *Information Processing and Management*, Vol. 11, pp. 39-57, 1975.
10. Hobbs, J. and R. Grishman, "The Automatic Transformational Analysis of English Sentences: An Implementation," to appear in *International Journal of Computer Mathematics*.
11. Sager, N. and R. Grishman, "The Restriction Language for Computer Grammars of Natural Language," *Communications of the ACM*, Vol. 18, pp. 390-400, 1975.

Design considerations of a database system in a clinical network environment

by SHI-KUO CHANG, M. O'BRIEN, J. READ, R. BOROVEC, W. H. CHENG and J. S. KE
University of Illinois
Chicago, Illinois

ABSTRACT

A database system designed in the context of a clinical network incorporating small computers, automated clinical instruments, and a variety of terminals is described. The database system is used for clinical data acquisition, research support, and clinical decision making support. The database system is based upon the relational approach, with important modifications to meet above design objectives and provide means for dynamic database reconfiguration. Its characteristics include: definition of elementary files as tables, a database skeleton describing the structure and contents of files in the database, a low-level database manipulation language based upon the Relational Algebra, and a frame-oriented user interface for high-level database manipulation.

INTRODUCTION

The Medical Information Systems Laboratory is developing a database system in the context of a clinical network incorporating small computers, automated clinical instruments, and a variety of terminals. The database system is intended to support patient care, clinical research, and research in advanced techniques of automated processing of medical information. The major support functions of the database system are summarized in Table I.

The primary functions of the clinical network include clinical knowledge acquisition, clinical research support, and clinical decision making support. Clinical data are collected from several sources: laboratory technicians, clerks, physicians, and automated clinical instruments. Clinical data should be checked for errors, formatted, and appropriately compressed, before being stored in the database. Moreover, database consistency and integrity should be maintained by the database system.

The clinical database is manipulated by local physicians for research purposes, using terminals connected to the clinical network. Another research use of the database is the sharing of clinical information via the Artificial Intelligence in Medicine (AIM) network. In the future, the clinical network could become a local node in a nationwide computer-communications network, so that outside physicians can conduct clinical research by gathering information from the databases at various local nodes.

The clinical database can also be used for clinical decision making. To aid clinical decision making, statistical information can be collected from the database, histograms can be plotted, using various mundane application programs. Data reduction and meaningful data manipulation capabilities are other useful features which can be provided by the database system. The database system should also facilitate the application of advanced artificial intelligence techniques such as automated inference and pattern recognition, so that such techniques can be explored as possible clinical decision making aids. Finally, since the clinical network will be used primarily by the Department of Ophthalmology of the Eye and Ear Infirmary, Uni-

TABLE I—Support Functions of the Database System

1. Clinical knowledge acquisition
   1.1 Data collection
       Technician Input
       Clerk Input
       Physician Input
       Automated Instruments
   1.2 Error control, data formatting and data compression
   1.3 Database consistency and integrity
2. Clinical research support
   2.1 Local physician research
   2.2 Outside physician research
   2.3 Artificial intelligence in medicine (AIM) research
3. Clinical decision making support
   3.1 Statistical information, graph plotting, and other mundane applications
   3.2 Data reduction
   3.3 Display and manipulation of stored information
   3.4 Automated inference aid
   3.5 Pattern recognition aid
   3.6 Graphics aid

versity of Illinois Medical Center, some interactive graphics capabilities are needed to present graphical information collected by some of the automated clinical instruments.

With the above objectives, we are designing a database system having the following characteristics:

1. Definition of elementary files as relational tables.
2. A database skeleton describing the structure and contents of files in the database
3. A low-level database manipulation language based upon Codd's Relational Algebra
4. A frame-oriented user interface for high-level database manipulation

In designing the clinical database system, we are heavily relying on the relational approach to database system design,[1-4] because we feel that the relational approach provides a unified, simple viewpoint to the user and at the same time also supports complicated deductive reasoning as exemplified by the first-order predicate calculus. The relational approach is also of interest from a methodological viewpoint, when we consider the problem of designing knowledge-based systems. Database systems are now being designed primarily for commercial applications in data processing. In order to extend the capabilities of database systems to support more sophisticated information processing applications such as automated inference, automated problem solving, and complex decision making, knowledge-based systems are needed. The relational approach offers possibilities of being extended to the design of knowledge-based systems.

The clinical database system is based upon the relational approach. However, important modifications are made to (a) meet the above mentioned design objectives, and (b) provide means for easy database reconfiguration, so that the database can be dynamically reconfigured for efficiency reasons. In what follows, the design of this database system will be described in some detail.

HARDWARE AND SOFTWARE ENVIRONMENT

The selection of hardware for the clinical network was based primarily on the availability of software tools for the development of the database system and other network control programs. The choice of Bell Laboratories UNIX operating system constrained our selection to PDP11-compatible machines. The clinical network includes a host computer, a number of satellites, a database module, and a number of terminals.

Host

The host system consists of a DEC PDP11/40 with 48K words of core memory, each word being two bytes long; two RK05 1.2 million word cartridge disk drives; two industry compatible magnetic tape drives; one Versatec electrostatic printer/plotter; and one high speed paper tape reader/punch which is used primarily for maintenance purposes. Communications with the satellites are via standard asynchronous serial lines. Initially these will operate at 9600 baud. As the final form of the clinical network takes shape, the lines and interfaces will be upgraded, and the old interfaces will be used for remote terminals.

Satellites

The four satellites are identical (at this time) and consist of a California Data Processor's 1/35 with 16K words of core memory, two serial I/O interfaces, and a bootstrap loader, mounted in a stand-alone cabinet which is converted to a desk when appropriate.

The choice of the CDP machine over a comparable DEC PDP11/35 was based primarily on price and flexibility. It is totally compatible with the DEC PDP11 and is based upon a fast, versatile microprocessor which will allow enrichment of the instruction set if necessary.

Each interface to clinical instruments will be a custom design tailored to make the data available in the most useful format. Preliminary specifications for each of the clinical interfaces have been completed. It is anticipated that some satellites (most notably the satellite for Glaucoma Clinic) will require more computer resources (more core memory and perhaps a local disk). These will be provided as the need arises.

Database module

The database module consists of front-end processor, a disk controller and disks. The processor and controller comprise a dual CDP 1/35. This configuration currently contains 32K words of core memory, memory management and an RK05 disk. Facilities for microprogramming consist of 256 words of alterable control store and a microconsole for debugging and maintenance.

The disks consist of four 80 megabyte Control Data Corporation 9762 Storage Modules giving a total storage capacity of 300 megabytes (after necessary formatting). The actual control of the disks is performed by a CDC CU/33 formatter which moves the heads, reads and writes the disks and performs error checking and correcting.

We plan to initially use the disks with a small interface which will make them look like DEC RP03 disks to the host machine. This will allow us to do most software development in the C language provided by the UNIX operating system.

Terminals

Three types of terminals are in use. The principal terminal in each clinic will be a modified Owens-Illinois

CVG-II plasma panel terminal with associated touch panel, both being outgrowths of the PLATO project.[5] Teleray 3700's and Infoton Vistar II's constitute the rest of the terminals for general use.

The system at this time (January, 1976) is partially operational. Six terminals are in use on the host system for database and operating system development. Links to one satellite and the Television Ophthalmoscope System[6] have been installed. The database module front-end is operational, and the disk components are being assembled.

In addition to the internal clinical network, a direct host/AIM link will be provided for access to the clinical database and for continued modelling of ophthalmic diseases, when AIM upgrades its 300 baud links to 1200 baud.

The selection of the UNIX operating system is made, primarily for the ease of software development under UNIX. The prototype database system is written in the above mentioned C language.[7] When the database module becomes operational, we also intend to run a stripped-down version of UNIX on the dual CDP 1/35. The lowest level database operations will later on be rewritten in microcode.

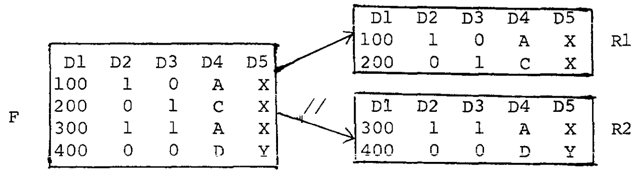
THE DATABASE

The clinical database comprises a collection of files. There are three types of files in the clinical database: *elementary files, composite files,* and *raw files.*

Elementary files

An elementary file is a *relational table** in Third Normal Form.[4] In the database, all composite files must be composed from elementary files. Since an elementary file is a relational table, all valid relational algebraic operations can be applied to an elementary file to extract the desired information. An elementary file is associated with a *descriptor set.* Each *descriptor* in the descriptor set has a name (usually a string of characters), a descriptor type, a format specification, and a descriptor domain. The descriptor types are as follows:

1. *Interval descriptor*—The descriptor domain is a linearly ordered set, with a minimum as well as a maximum element.
2. *Nominal descriptor*—The descriptor domain is an unordered set. Descriptors with values as character strings are regarded as nominal descriptors. So are logical descriptors which take on the logical values 0 or 1. However, there is usually an implicit collating sequence for character strings, so that they still can be sorted into a collating sequence.
3. *Structural descriptor*—The descriptor domain has a specified mathematical structure, such as a tree, a lattice, a semi-lattice, a directed graph, etc.



F

D1	D2	D3	D4	D5
100	1	0	A	X
200	0	1	C	X
300	1	1	A	X
400	0	0	D	Y

R1

D1	D2	D3	D4	D5
100	1	0	A	X
200	0	1	C	X

R2

D1	D2	D3	D4	D5
300	1	1	A	X
400	0	0	D	Y

Figure 1—Vertical concatenation

The format of a descriptor is usually specified by a *picture* of the format, such as X(3), 9(5), which mean a character string of length 3 and a numeric string of length 5, respectively. In the current implementation, only the first two types of descriptors are allowed.

An elementary file consists of a number of records (or rows, tuples), each record being of the form $(d_1, d_2, \ldots, d_n)$, where d_i belongs to the descriptor domain of descriptor D_i.

Composite files

A composite file is a relational table which is defined in terms of other relational tables using certain *composition rules.* As an example, R1 and R2 are two constituent files. A composite file F can be defined as the *vertical concatenation* of R1 and R2, or

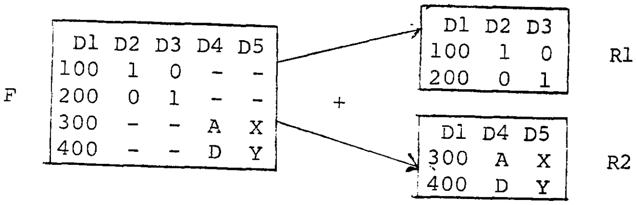
$$F=R1//R2$$

where // is the vertical concatenation operator, provided that R1 and R2 have similar descriptor sets, as illustrated in Figure 1. Similarly, a composite file F can be defined as the *union* of two constituent files R1 and R2, or

$$F=R1+R2$$

where + is the union operator, provided that R1 and R2 have similar concatenated keys, as illustrated in Figure 2, where D1 is the common key, and – denotes an undefined entry in the relational table.

A *concatenated key* is a subset of the descriptor set, which takes on unique values for each record in a file. If the concatenated key comprises one descriptor, that



F

D1	D2	D3	D4	D5
100	1	0	-	-
200	0	1	-	-
300	-	-	A	X
400	-	-	D	Y

R1

D1	D2	D3
100	1	0
200	0	1

R2

D1	D4	D5
300	A	X
400	D	Y

Figure 2—Union

descriptor is called a *key,* such as D1 for file F in Figure 2.

A third composition rule is *qualified vertical concatenation,* which can be used to combine files by adding an extra descriptor to indicate the individual identities of constituent files. An example is illustrated in Figure 3, where $F denotes the new descriptor for identifying constituent files. The composite file F is the qualified vertical concatenation of R1 and R2, or

F=R1(//$F)R2

The above example illustrates the composition of F from R1 and R2. Conversely, F can also be decomposed into several smaller files, as illustrated in Figure 4, where the decomposition is made with respect to descriptor D1, and

F=$F1(//D1)$F2(//D1)$F3

where $F1, $F2 and $F3 are new names for the constituent files.

Similarly, another composition rule is *qualified union,* which can be used to combine files with different descriptor sets by adding an extra descriptor to indicate the identities of the constituent files.

The last composition rule is the *horizontal concatenation* rule. A composite file F can be defined as the horizontal concatenation of constituent files R1 and R2, or

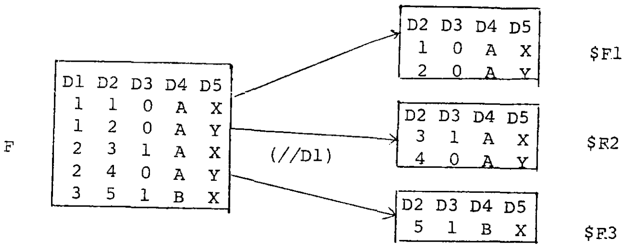
F=R1(*D1)R2

by equijoining the two relational tables on descriptor D1, where * denotes the *equijoin operator,* and D1 must be the key for both R1 and R2, as illustrated in Figure 5.

In horizontal concatenation, the constituent files can be joined on any common concatenated key, in which case the concatenation operation is written as (*D1,D2,...,Dn), where {D1,...,Dn} is the common concatenated key.

Raw files

Raw files are files not in the form of relational tables. In clinical data collection, data entered by human users could be appended to relational tables which have already been defined by the users previously. Some data collected by clinical instruments, however, will not be suitable to be stored in relational tabular form directly. Graphical information such as visual field charts, pictorial information such as those provided by the Television Ophthalmoscope System (the TVO System), are examples of nonrelational data. It will be wasteful to store such information in relational tables. Moreover, in the intended applications, such graphical and pictorial data will invariably be processed by special application programs. Therefore, they should be stored in raw files, with no restrictions on format, record length and file size.



F

D1	D2	D3	D4	D5
1	1	0	A	X
1	2	0	A	Y
2	3	1	A	X
2	4	0	A	Y
3	5	1	B	X

$F1

D2	D3	D4	D5
1	0	A	X
2	0	A	Y

$F2

D2	D3	D4	D5
3	1	A	X
4	0	A	Y

$F3

D2	D3	D4	D5
5	1	B	X

Figure 4—Decomposition of F

Raw files can also be used for input/output purposes. After an elementary file has been defined, the following statement

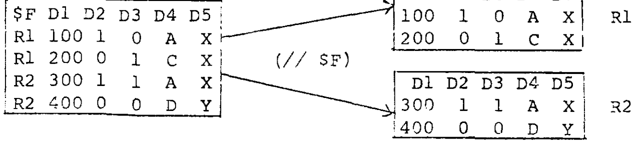
READ R1 FROM U1

can be used to transform a raw file U1 into an elementary file R1 in relational tabular form. Similarly, the statement

WRITE R1 TO U1

will transform R1 into a nonrelational format and store it in U1. If the phrase "FROM U1" is omitted the standard input file is assumed, which is normally taken to be the user terminal. Similarly, the omission of "TO U1" causes output to be directed to the standard output file, and the output normally appears at the user terminal.

The above input/output mechanism is very useful for the clinical research applications. For example, a file in relational tabular form may first be transformed into a raw file. A data reduction program based upon Variable-valued Logic Reduction[5] can then be applied to the raw file. Finally, the resultant raw file can again



$F	D1	D2	D3	D4	D5
R1	100	1	0	A	X
R1	200	0	1	C	X
R2	300	1	1	A	X
R2	400	0	0	D	Y

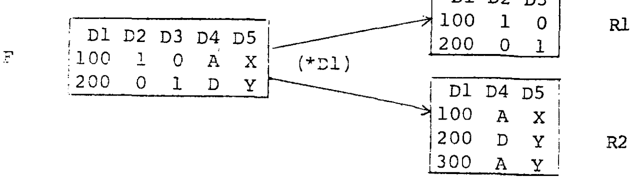
R1

D1	D2	D3	D4	D5
100	1	0	A	X
200	0	1	C	X

R2

D1	D2	D3	D4	D5
300	1	1	A	X
400	0	0	D	Y

Figure 3—Qualified vertical concatenation



F

D1	D2	D3	D4	D5
100	1	0	A	X
200	0	1	D	Y

R1

D1	D2	D3
100	1	0
200	0	1

R2

D1	D4	D5
100	A	X
200	D	Y
300	A	Y

Figure 5—Horizontal concatenation

be recast into relational tabular form. Similarly, picture processing applications are also applied to raw files, with subsequent transformation into relational tables, after extensive preprocessing has been made.

A final use of raw files is to *equate* a raw file to an elementary file (see later section). Any database operations to be performed on the elementary file will be actually performed by a special program associated with the corresponding raw file. For example, file U2 is a raw file with an inverted file structure. The associated special program is in a UNIX file P2. File U2 is equated to an elementary file R2. Whenever a database operation is to be performed on R2, the operation command and arguments are passed to P2, which then performs some operation on U2. In this way, files with different file structures could be treated as part of the relational database and at the same time still retain their idiosyncratic structures. It is thus unnecessary to transform all files into relational tubular form, which may not be economical either in storage or in speed, or both.

DATABASE SKELETON

With the above described three types of files—elementary files, composite files, and raw files—a database can be structurally characterized by a hierarchical schema, called a *database skeleton*. Here we will restrict our attention to one user's view of the database. Multiple user's views can be accommodated by having multiple database skeletons. An example is shown in Figure 6.

In the database skeleton, U1, U2, U3 and U4 are raw files. U1, U2 and U4 have associated application programs P1, P2, and P4. The composite file F1 is defined as the equijoin on D1 of two elementary files R1 and R2. The composite file F2 is defined as the vertical concatenation of an elementary file R3, and another composite file F3, which in turn is the equijoin on D2 of three elementary files R4, R5 and R6. The elementary file F4 is equated to raw file U4, whose associated program is P4.

From the user's viewpoint, he need only be aware of the existence of U1, U2, U3, F1, F2 and F4. The structures below composite files could be opaque to him.

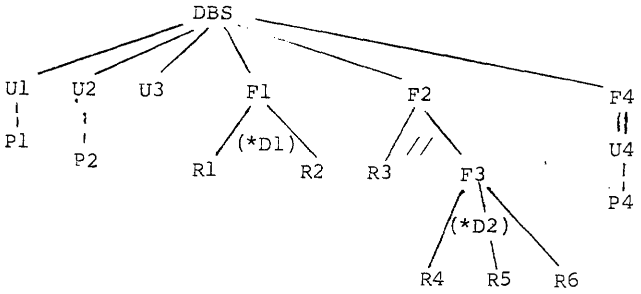


Figure 6—A database skeleton

He also may not know that F4 is equated to U4, and the relational table is in reality implemented as an inverted file.

Every file in the database skeleton also has an associated logical expression called the file *characteristics*, which characterizes the contents of this file. For composite files and elementary files, every record in a file must satisfy the characteristics logical expression. For raw files, the characteristics clause can be used to register typical samples, or prototype records, of a raw file.

A more detailed discussion of the database skeleton, together with its formal model and some theoretical results, can be found in Reference 9. Intuitively, the database skeleton serves the function of a structured file directory, similar to UNIX's tree directory. However, it is also a model of the database, with information on file size, file contents, and database operation statistics. Such information could be used (a) to optimize the performance of database operations, and (b) to dynamically restructure the database to improve system performance. These topics are also treated in Reference 9.

To illustrate how the database skeleton can be used for efficient searching, consider the case of a composite file F which is the vertical concatenation of R1 with the characteristics $(D1 \leq 200)$, and R2 with the characteristics $(D1 > 200)$. If it is intended to find all records in file F with $D1 \geq 300$, then the constituent file R1 need not be searched, because no record in R1 satisfies the search condition. This is discovered when the expression $(D \leq 200) \wedge (D1 > 300)$ is reduced to a logical zero. Therefore, a structurally organized database skeleton could help reduce database processing time.

DATABASE MANIPULATION LANGUAGE

The database manipulation language is based upon Codd's Relational Algebra.[3] An algebraic language is chosen for two reasons. First of all, an algebraic language is flexible and can easily be extended. The language is easily embedded in a host language, in our case the C language, so that the powerful instruction set of the host language can be used to write application programs or to design flexible user interfaces. In fact, the database manipulation language is considered to be a low-level language for internal use. The end user will communicate with the database system via the user interface described later.

The second reason for choosing an algebraic language is that it permits easy definition of structurally composed files. Although relational tables are useful as a unifying viewpoint for the end user, in implementing a realistic database system on a small computer, performance considerations cannot be overlooked. When a database grows larger and larger, the efficient storage and retrieval of possibly very large files becomes the primary concern of the system designer. In such cases, it is natural to consider breaking up or decomposing a large file into smaller pieces. The algebraic database manipulation language can be used to define structurally composed files and to dynamically restructure the files, so that storage requirements and processing time can be reduced.

The basic commands** of the database manipulation language are summarized in the Appendix. Several examples will be given in this section. The vertical concatenation of files, as illustrated in Figure 1, can be defined as follows.

```
DEFINE CFILE F(D1,D2,D3,D4,D5)
    STRUCTURE (//)
DEFINE EFILE R1(D1,D2,D3,D4,D5)
DEFINE EFILE R2(D1,D2,D3,D4,D5)
INCLUDE (R1,R2) IN CFILE F
```

Similarly, the horizontal concatenation of files, as illustrated in Figure 5, can be defined as follows.

```
DEFINE CFILE F(D1,D2,D3,D4,D5)
    STRUCTURE (*D1)
DEFINE EFILE R1(D1,D2,D3)
DEFINE EFILE R2(D1,D4,D5)
INCLUDE (R1,R2) IN CFILE F
```

In the definition of elementary or composite files, a *characteristics* clause can be included to describe concisely the contents of this file. Access and update can also be controlled by other clauses, as illustrated in the following example.

```
DEFINE EFILE R1(D1,D2,D3)
    CHARACTERISTICS (D1≤200)
    RETRIEVAL (D4='X')
    DELETION (USERID='SMITH')
    MODIFICATION (USERID='SMITH')
```

The above example provides the following information: name of elementary file is R1, with descriptors D1, D2, and D3; every record in R1 has D1 less than or equal to 200; all records with D3 equal to 'X' can be retrieved by any user; only user allowed to delete and/or modify records is 'SMITH'. (USERID is a global keyword. Other such keywords are PASSWORD, GROUPID, CLINICID, etc.)

A composite file can be dynamically restructured, as illustrated in the following example.

```
DEFINE CFILE F(D1,D2,D3)
    STRUCTURE (+)
DEFINE EFILE R1(D1,D2,D3)
DEFINE EFILE R2(D1,D2)
INCLUDE (R1,R2) IN CFILE F
.....
DELETE R2
.....
DEFINE EFILE R3(D1,D2)
DEFINE RFILE U3(P3) EQUAL TO EFILE R3
INCLUDE (R3) IN CFILE F
. . . . .
```

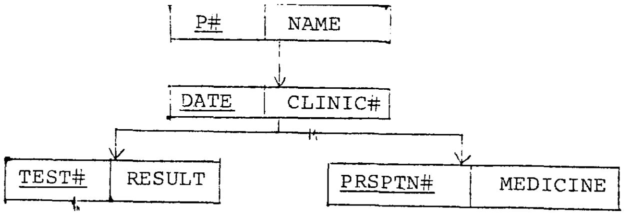


Figure 7—Hierarchical patient record structure

In the above example, F originally is the union of R1 and R2. Later on, R2 is removed from F. A *virtual* elementary file R3 which is equated to a raw file U3, is added to F. Thus, F is the union of R1 and R3 (which is really U3). U3 might have a file structure more efficient for certain retrieval operations, so that database operations performed on F can be processed more efficiently.

Hierarchical record structures, such as those used in IMS,[10] occur quite frequently in practice. Hierarchical structures can easily be represented using composite files and the horizontal concatenation rule, as illustrated in the following example. The hierarchical patient record structure is shown in Figure 7.

In Figure 7, the underlined fields are the key fields. This hierarchical patient record can be defined in our database as in Table II.

Suppose a composite file Fi has descriptor set Si and concatenated key Ki for horizontal concatenation. Suppose its constituent composite file Fj has descriptor set Sj and concatenated key Kj for horizontal concatenation. Then the following relation must hold,

$$Kj \subset Ki \subset Sj \subset Si$$

As before, a hierarchically structured composite file can also be dynamically restructured.

It can be seen that the sub-partitions of a composite file (or conceptually, the sub-regions of a two-dimensional relational table) can be assigned different protection conditions. Therefore, if part of the patient record is confidential and should never be made public, that portion could be separated out as a constituent file and given the most severe protection condition. This confidential file can also be stored in a separate

TABLE II—Defining a Hierarchical Patient Record

```
DEFINE CFILE F(P#,NAME,DATE,CLINIC#,TEST#,
   RESULT,PRSPTN#,MEDICINE) STRUCTURE (*P#)
DEFINE EFILE R1(P#,NAME)
DEFINE  CFILE  G(P#,DATE,CLINIC#,TEST#,RESULT,
   PRSPTN#,MEDICINE) STRUCTURE (*P#,DATE)
INCLUDE (R1,G) IN CFILE F
DEFINE EFILE R2(P#,DATE,CLINIC#)
DEFINE EFILE R3(P#,DATE,TEST#,RESULT)
DEFINE EFILE R4(P#,DATE,PRSPTN#,MEDICINE)
INCLUDE (R2,R3,R4) IN CFILE G
```

storage volume with tight security control. Thus, the above described structured database protection scheme can be very useful to the database manager in a clinical network environment.

USER INTERFACE

The user interface is frame-oriented, with emphasis on tabular presentation of structured information to the end user. This approach is based upon the frame-oriented approach of the PLATO system.[5] The CVG-II touch panel plasma display terminal is an ideal vehicle for designing such frame-oriented user interface.

There are three modes provided by the user interface. As the user starts the interactive session, he is presented with the first frame, which displays the three modes: the *data entry mode*, the *database manipulation mode*, and the *table manipulation mode*. In addition to the three basic modes, in every frame there is a *help* button. If the user touches the help button, self-explanatory information concerning the user interface itself, the database manipulation language, or the database system will be provided. There is also a *dbs* button. If the user touches the *dbs* button, he can browse through the database skeleton, which can be regarded as an enhanced database directory. Other buttons enable the user to *back up* to the previous frame, *exit* to the starting frame, or *move on* to the next frame.

In the *data entry mode*, the user first selects a file name. The user interface will prompt the user to enter a record, and then send this record to the database system to append to that file. For each record, fields will be checked against the correct format, and default values will be inserted in the unspecified fields. If a descriptor has only a limited number of possible values, such as M or F for the SEX descriptor, these values will be displayed on the screen to allow the user to make a selection.

In the *database manipulation mode*, the user can enter a retrieval/update command by selecting the appropriate files, descriptors, conditions, and operations, which are provided by the user interface. The user interface will translate user's specification into low-level commands of the database manipulation language.

For example, suppose the patient record is hierarchically structured, as shown in Figure 7. The patient file F is a composite file, composed from four elementary files, as illustrated in Table II. The user may wish to find the names of patients who visited Clinic #12. Using the user interface, he makes the following selections:

```
FILENAME: F
DESCRIPTORS TO BE RETRIEVED: NAME
CONDITIONS: *CLINIC# == '12'*
```

The user interface will translate the above specifications into the following commands, with the aid of the database skeleton:

```
T1=R2( *CLINIC# == '12'* )
T2=T1(P#,DATE,CLINIC#) (*P#)
    R1(P#,NAME)
T3=T2(NAME)
PRINT T3
DELETE T1,T2,T3
```

It is obvious that the user interface could attempt to optimize the translated commands by using such information as the size of the constituent files, approximate processing times for various operations, and the structure of the files. These problems are considered in Reference 9.

In the *table manipulation mode*, the user first selects a relational table, such as the patient file F. He can then browse through this big table. He can declare certain descriptors to be *stationary*, such as P#, whose column will always remain on the screen. He can declare certain descriptors to be *movable*, whose columns will be moved in and out the screen. He can also *drop* unwanted descriptors. There are touch buttons allowing him to *scroll left, scroll right, scroll up,* and *scroll down,* positioning the viewing window anywhere in the big table. There is also a *single record* option, allowing the user to view the file in a record-by-record fashion.

In the table manipulation mode, the user can only make projections and restrictions. Other more time consuming database manipulations will not be made available to him. We believe that projections and restrictions are by far the most useful relational algebraic operations. Other relational algebraic operations are too expensive to be provided in the table manipulation mode.

After having selected a desired sub-table, the user can store it as a temporary raw file using the *store* button. Special application programs can then be invoked to *plot* diagrams, compile *histograms,* and perform *data reduction* operations. (The italicized words are touch buttons defined as fixed areas on the touch panel.)

IMPLEMENTATION STATUS

In the preceding sections, the design of a database system in the context of a clinical network has been described in some detail. Currently, a prototype database system has been implemented. The software was developed in the UNIX environment, using the C language. The prototype database system consists of the following:

1. *User interface*—It supports the three modes described above, with the table manipulation programs still under development.
2. *RAIN*—A Relational Algebraic INterpreter was

implemented which accepts and executes commands in the database manipulation language, also called RAIN.

3. *Data compression programs*—Several data compression programs are available to reduce input data into compressed internal format, and to perform corresponding inverse transformation. As far as the end user is concerned, he only sees data in the user-specified external format.
4. *VVL data reduction program*—A data reduction program based upon VVL reduction techniques has been written, which can be invoked to reduce a given relational table into a disjunction of VVL logical expressions. These logical expressions can be evaluated by the user, and used as the characteristic expression describing the contents of a relational table.

The current implementation, when completed, will be used as an experimental tool for trial usage. The above described database system is a first step toward the design of a database system which can support mundane applications as well as certain more esoteric research applications. The system will undoubtedly undergo major modifications and evolution. However, areas for the application of optimization techniques have been identified and are under investigation. Such database systems which are implemented on small computers and still support advanced research applications could find wide range applications involving sophisticated decision making. The clinical network environment provides an interesting test bed for the design of such systems.

REFERENCES

1. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM*, Vol. 13, No. 6, 1970, pp. 377-387.
2. Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," *Proc. of ACM-SIGFIDET Workshop on Data Description, Access and Control*, San Diego, 1971.
3. Codd, E. F., "Relational Completeness of Data Base Sublanguages," in Courant Computer Science Symposia, No. 6, *Data Base Systems*, New York, May 1971.
4. Codd, E. F., "Further Normalization of the Data Base Relational Model," in Courant Computer Science Symposia, No. 6, *Data Base Systems*, New York, May 1971.
5. Bitzer, D. L. and J. A. Easley, "PLATO: A Computer-Controlled Teaching System," in *Computer Augmentation of Human Reasoning*, Edited by M. A. Sass and W. D. Wilkinson, Spartan Books, 1965, 90-103.
6. McCormick, B. H., R. T. Borovec, J. S. Read and R. C. Amendola, "Image Processor for Biomedical Research," in *Computers in Life Science Research*, edited by W. Siler and D. A. B. Lindberg, Plenum Press, 1975, pp. 129-135.
7. Ritchie, D. M. and K. Thompson, "The UNIX Time-Sharing System," *Comm. ACM*, Vol. 17, No. 7, 1974, pp. 365-375.
8. Michalski, R. S., "AQVAL/1-Computer Implementation of a Variable-Valued Logic System VL_1 and Examples of Its Application to Pattern Recognition," *Proc. of First International Joint Conference on Pattern Recognition*, Washington, D.C., October 1973, pp. 3-17.
9. Chang, S. K., "Database Skeleton—A Formal Model," Technical Report, Medical Information Systems Laboratory, Department of Information Engineering, University of Illinois at Chicago Circle, Chicago 1976.
10. *Information Management System/360, Version 2, General Information Manual*, GH20-0765-4, IBM, June 1973.

APPENDIX—DATABASE MANIPULATION LANGUAGE

A. *File definition and file input output*

A1. DEFINE EFILE R (P1)
(D1(9(2),n1),D2(9(3),n2),D3(X(5),n3))

An elementary file R is defined, and its definition is stored in the database skeleton. The file R is still empty. P1 is name of data compression/conversion program to convert data from external format into a condensed internal format, and vice versa. D1, D2, D3 are descriptors. Their external formats are specified by 9(2), 9(3), and X(5). The integers n1, n2 and n3 refer to conversion rules in P1. If there is no data compression (the external format is the same as the internal format), then P1, n1, n2, and n3 may be omitted.

A2. DEFINE CFILE F(P2)
(D1(9(2),n1),D2(9(3),n2),D3(X(5),n3))
STRUCTURE (//)
CHARACTERISTICS (D1≤200)
RETRIEVAL (D4='X')
DELETION (USERID='SMITH')
MODIFICATION (USERID='SMITH')

A composite file F is defined. Its definition, structure, characteristics and other protection conditions are stored in the database skeleton. The file F is still empty.

A3. INCLUDE (R1, R2, R3) IN CFILE F

The files R1, R2, and R3, which can be either elementary files or composite fiiles, are included as constituent files of the composite file F. The structural links are stored in the database skeleton.

A4. DEFINE RFILE U(P3) EQUAL TO EFILE R

A raw file is defined, and its definition is stored in the directory. The raw file U can be associated with an application program P3, which is a UNIX file. If the clause "EQUAL to R" is added, then the raw file U is equated to a relational table R. All database operations on R will be relegated to P3.

A5. EFILE R

(D1='1',D2='3',D3='4',D4='A')
(D1='3',D2='0',D3='0',D4='B')
&

When the elementary file R is empty, it is initialized by directly specifying its contents. If a certain descriptor is omitted in the specification, its value is undefined. The database skeleton is modified by entering the record number of R and setting a pointer to the first location of the file. When the elementary file R is not empty, the new records are appended to the end of the existing file.

A6. READ $\begin{bmatrix} \text{E} \\ \text{C} \\ \text{N} \end{bmatrix}$ R FROM U1

The elementary file R is initialized by reading in the records from a raw file U1. The database skeleton is modified by entering the record number of R and setting a pointer to the first location of the file R. The conversion code E indicates that the raw file U1 contains data in external format (as specified in the DEFINE statement), C indicates that U1 contains data already in condensed internal format, and N indicates conversion of binary integers. If the clause "FROM U1" is omitted, standard input is assumed.

A7. WRITE $\begin{bmatrix} \text{E} \\ \text{C} \\ \text{N} \end{bmatrix}$ R TO U1

The file R is written to a raw file U1, according to conversion specifications. If the clause "TO U1" is omitted, the standard output is assumed.

B. *Maintenance operations*

B1. ERASE R
The file is deleted from the database skeleton. If a composite file is deleted, all its constituent files are also deleted.

B2. RENAME R1 R2
The name of R1 is changed to R2, and the database skeleton is updated.

B3. DISPLAY F
The format, structure, characteristics, protection conditions and constituent files of F are printed via standard output. If the command is "DISPLAY ALL", the contents of the database skeleton are printed.

B4. PRINT F
File F is printed via standard output, using external format.

C. *Basic data operations*

C1. INSERT (D1='2',D3='0') INTO R1
A record with D1 equal to 2 and D3 equal to 0 is inserted into file R1, which must be either a composite file or an elementary file.

C2. DELETE (<condition>) FROM R1
All records in R1 satisfying given <condition>‡ are deleted.

C3. MODIFY (<condition>) OF R1 TO (D1='2',D3='0')
All records in R1 satisfying given <condition> are modified by changing the value of D1 to '2', and the value of D3 to '0'. Other descriptor values are unchanged.

C4. RETRIEVE (<condition>) of R1 (D1,D2) INTO R2
Records in R1 satisfying the given ⟨condition⟩ are retrieved into a file R2. R2 will have the descriptor set {D1,D2}, and other descriptors of R1 are discarded. If R2 is an existing file, it is redefined.

D. *Relational algebraic operations*

D1. (PROJECTION) R1=R2(D1,D2)
A file R1 with descriptors D1, D2 is created, which is the projection of R2 on D1, D2. If R1 is an existing file, it is redefined.

D2. (RESTRICTION) R1=R2 (<condition>)
A file R1 having the same descriptor set as R2 is created, which is the restriction of R2, so that only records satisfying the given <condition> are in R1. If R1 is an existing file, it is redefined.

D3. (JOIN) R1=R2(D1,D2) (*D2) R3(D2, D3)
R1 is the equijoin of R2(D1,D2) and R3(D2, D3) on key D2. R1 has descriptor set {D1, D2, D3}. If R1 is an existing file, it is redefined.

D4. (DIVISION) R1=R2(D1,D2) (/D2) R3 (D2)
R1 is the division of R2(D1,D2) by R3(D2) on D2. R1 has descriptor set {D1}. If R1 is an existing file, it is redefined.

D5. (VERTICAL CONCATENATION) R1=R2 (D1,D2)//R3(D1,D2)
R1 is the vertical concatenation of R2 and R3. R1 has same descriptor set as R2 or R3. If R1 is an existing file, it is redefined.

D6. (DIFFERENCE) R1=R2(D1,D2)−R3,(D1,D2)

R1 is the set-theoretic difference of R2 and R3. R1 has same descriptor set as R2 or R3. If R1 is an existing file, it is redefined.

D7. (PRODUCT) R1=R2(D1,D2) ** R3(D3,D4)

R1 is the Cartesian product of R2 and R3. R1 has descriptor set {D1,D2,D3,D4}. If R2 and R3 contain same descriptor Di, R1 will contain Di and $Di. If R1 is an existing file, it is redefined.

D8. (INTERSECTION) R1=R2(D1,D2) * R3 (D1,D2)

R1 is the set-theoretic intersection of R2 and R3. R1 has same descriptor set as R2 or R3. If R1 is an existing file, it is redefined.

D9. (UNION) R1=R2(D1,D2)+R3(D1,D3)

R1 is the set-theoretic union of R2 and R3. The descriptor set of R1 is union of that of R2 and R3, provided that R2 and R3 have common concatenated key. If R1 is an existing file, it is redefined.

E. *Miscellaneous*

E1. END $\begin{bmatrix}\text{SAVE}\\ \text{NOSAVE}\end{bmatrix}$

Notes

* We will use "relational table" or "table" to refer to the physical realization of a mathematical relation.

** Any command in the database manipulation language can be split into several lines, provided that a back-slash (\) is entered immediately preceding the new-line or return key.

‡ A condition is a Boolean expression of terms, where each term is either of the form <descriptor> <rel-op><literal>, or of the form <descriptor> <rel-op><descriptor>. <rel-op> is any one of the relational operators: ==(equal), != (not equal), > (greater than), >= (greater or equal), < (less than), <= (less or equal). A logical "and" is written as &&, and a logical "or" is written as ||. The logical expression is surrounded by two asterisks.

Correct problem statements in biomedical data processing

by N. I. MOISEEVA, M. YU. SIMONOV and V. M. SYSUEV
Academy of the Medical Sciences of the USSR
Leningrad, USSR

ABSTRACT

The correct formulation of any task appears to be not only the first step in its solution but sometimes the solution itself. Biomedical data processing is unlike other problems encountered by biologists and they often use mathematical methods without taking into consideration both the possibilities and limitations of the given methodology and the properties of the data to be processed. As a result the approach to the solution of the problem is often inadequate.[1]

Moreover, biologists encounter difficulties with the choice of a method for data conversion into the form suitable for mathematical processing. Often this leads to the incomplete utilization of the power of a given mathematical method. Sometimes only the correct data representation is sufficient to draw final conclusions. For example, data given in the form of an interval distribution histogram or mean frequency function allow the investigator to obtain information on time-dependent features of the process.

Science "creates and supports conditions whereby functional foundations become the field of controversy, resulting in competitive but different ways of doing the same thing. In other words the number of alternatives constantly increases due to science."[2]

In fact the problem of choice always faces the investigator not only in relation to the optimal method of data processing, but also in relation to optimally configuring electronic devices to be used in the implementation of these methods.

The correctly formulated requirements for biomedical data processing could be and should be used as a basis for this choice. Thus there arises the special problem of stating correctly the processing task itself. From our point of view the correct approach to its solution must take into account all a priori information. We must systematically look at the final aims of our research (in the sense that it is a biological data processing task) and at existing methods of analysis, their scope and limitations.

PECULIARITIES IN THE STATEMENT OF BIOMEDICAL TASKS

The complications arising in the study of biological objects derive above all from the fact that life itself, from the point of view of physics, "is too intricate and does not lend itself to mathematical interpretation."[3] Living organisms possess a large number of possible stable states; every concrete subject during its whole life uses only some of them, and different subjects prefer different approaches (if it is possible to use this word in reference to processes in the vegetatic nervous system). Moreover, in the same organism even stable states are different as they depend, for example, on age and functional states. Thus the problem of selecting appropriate statistical tools in biology becomes quite intricate. In fact, even the idea of "normality" is not clearly defined in biology so far.[4]

Transmission of signals in living organisms is carried on both with the aid of discrete impulses (e.g., in the nervous system) and analog-type information, like the constant potential of membranes and tissue, humoral regulation of functions and so on. Under these conditions signals propagate in two ways—along a specific channel to a specific organ or group of cells and "in a general way," without any defined "address." The latter signals could act upon any part of the organism and could, in fact, be perceived either by any part of the organism (i.e., general nonspecific reactions like arousal) or by the tissues and organs specially prepared for the reception of this particular signal or group of signals. For example, the hormones of endocrinous glands excreted in blood are intended for specific "target-tissue."

The next complication arises from the fact that there are no really stable functions in a living organism, every one of them representing oscillation processes.

The relative stability of an organism is the result of the rhythmic activity of the regulating systems. Some of these rhythms are endogenous: That is, they originate in the organism itself. Others reflect external events like the change of day and night, seasons of the year, and other influences from the environment.

As a specific property of this oscillating system the irregularity of growth and decay of different parts should be mentioned. For example, the growth of bones and muscles in human beings practically ceases as we attain adulthood, but some internal organs and the cardio-vascular system continue their development during the whole life. Organisms develop and degrade in an irregular manner.

All this leads to the idea that biological analysis must take into account structural (synchronous) as well as temporal (diachronic, functional) aspects of living objects.

SUGGESTED CLASSIFICATION OF BIOMEDICAL DATA PROCESSING TASKS.

In the literature on this problem we were able to find only one attempt to systematize biomedical tasks. We are here not concerned with the classification based on the coefficient of utilization of computer memory.[5] Rather we refer to the paper by Dixon[6] in which the author discusses both the so-called "source of information" (in fact, dealing with the modelling of functions, selecting interconnections, estimating the state of health etc.) and the desired processes. The taxonomy is rather incidental and classification indicators are not selected.

We made an attempt to distinguish two groups of biomedical processing tasks after having selected as a basic indicator the relation to time: those tasks where some variables and their interconnections are analyzed without regard to time (synchronous analysis) and those tasks where the processes develop in time (diachronic analysis).

Within each of these groups the tasks are classified according to final aims (description and identification or classification) on one side and to the object of the investigation (elementary events, processes or interrelations) on the other side.

In Tables I and II the systematization of biomedical processing tasks is shown both for the synchronous and diachronic analysis.

It is well-known that all the curiosity as to events in our universe can be expressed with the help of only six questions: who (what); what kind of; where; when; how; why (what for). All entries in Table I are in response to questions about the kind of objects and the nature of their interactions; or to questions whether certain objects belong to certain classes. Entries in Table II address the questions "what kind of" and "when" (analysis of elementary events and their interactions within the time continuum), the questions "what kind of", "when" and "where" (analysis of processes and their interconnections), or the questions "what" and "when" (classification of phenomena and processes). The questions "how" and "why" are not within the scope of our analysis.

Definition of causes, aims and mechanisms is derived from the combination of analytical and synthetical processes in the human brain. Without the latter human component the results of analysis bring to life only the question "so what?".

SUGGESTED CLASSIFICATION OF BIOMEDICAL DATA ANALYSIS METHODS

Many books on mathematics for biologists exist, of course, but there are no specific methodologies (and, of course, branches of mathematics) specially tailored to deal with biomedical problems. Biologists in their investigations simply use more or less successfully "standard" mathematics. We were unable to find in the literature any guidelines for biologists to aid their understanding of the relations between different methods or giving advice on the use of various kinds of processing techniques.

Biologists and non-biologists are becoming aware of the need to systematize methods of analysis. Isolated attempts to create particular schemes of analysis have been made, as shown in Tables III and IV (from the

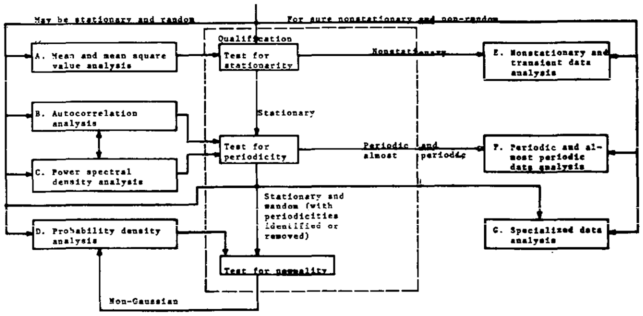
TABLE I—Levels of Structural (Synchronous) Analysis of Biological Objects

	Analysis of structural units within the system		Analysis of the interconnections between the structural units	
	Analysis of separate elements	Analysis of a complex of elements	Analysis of interconnections between the separate elements	Analysis of interconnections between the complexes of elements
Description and identification of objects	Description of indicators of separate elements and the analysis of their informativity	Description of characteristics of a complex of elements as a whole	Description of parameters, general features of the interconnections between the elements	Description of the parameters and general features of interconnections between the systems, classes, taxones, clusters, etc.
Classification of objects	Classification of elements, construction of classes, taxones, clusters, etc	Classification of the complexes of elements (systems, classes, etc.)	Classification of interconnections between the elements, separation of the systemic ones, extraction of groups of elements which form the system	Classification of interconnections between the systems, classes, and separation of the systemic interconnections of the second order

TABLE II—Levels of the Diachronic Analysis of Biological Objects

	Analysis of processes which are set of separate events and have initial and final points		Analysis of processes reflecting the continuous change of element state where the initial and final points can be selected only arbitrarily	
	Analysis of elementary events	Analysis of interconnections between the events	Analysis of processes by themselves	Analysis of interconnections between the processes
Description and Identification	Description of the events by means of time series and evaluation of their parameters	Discovery and description of type and the degree of interconnections between separate events	Description of processes by means of: a) representation in a form of a sequence of momentary states b) representation in the form of a sequence of events c) search of the repeated states and regular changes	Definition of the type and parameters of the interconnections between the processes
Classification	Separation of classes of elementary events	Selection of the causal-consequence interconnections, groups of events forming the phenomenon	Selection of the classes of processes -- relaxational, implulse, hormonic, etc.	Selection of the deterministic and stochastic causal-consequence interconnections

TABLE III—General Procedure for Analyzing Individual Sample Records


May be stationary and random
For sure nonstationary and non-random
Qualification
A. Mean and mean square value analysis
Test for stationarity
Nonstationary
E. Nonstationary and transient data analysis
B. Autocorrelation analysis
Stationary
Test for periodicity
Periodic and almost periodic
F. Periodic and almost periodic data analysis
C. Power spectral density analysis
Stationary and random (with periodicities identified or removed)
D. Probability density analysis
G. Specialized data analysis
Test for normality
Non-Gaussian

book by Bendat and Piersol) for the complex of realization and Table V for time series used for the rhythmometric investigations.

Since neither general schemes nor a classification of methods exist, we made an attempt to classify the methods of analysis on the basis of the above taxonomy of the data processing tasks.

As time and spatial coordinates are processed identically in calculating procedures we combined the methods of spatial structural analysis and methods of time series analysis.

The basis for our classification is the ability of a given method to answer one or more questions from the above mentioned set. Thus we systematized ways of processing and separated the methods which could be used as:

—methods of qualitative analysis (indicative methods) answering questions on the general features of the object, event or a process (from a mathematical point of view).

—quantitative methods of investigation for estimation of the numerical characteristics of processes or objects.

—methods for qualitative and/or numerical description of interconnections.

—methods for analysis of external (related to the environment) features of objects and processes.

TABLE IV—General Procedure for Analyzing a Collection of Sample Records

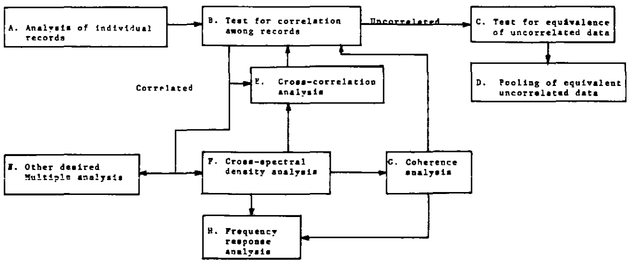
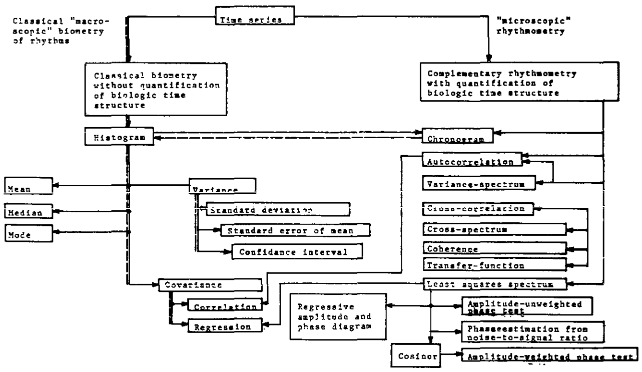

A. Analysis of individual records
B. Test for correlation among records
Uncorrelated
C. Test for equivalence of uncorrelated data
Correlated
E. Cross-correlation analysis
D. Pooling of equivalent uncorrelated data
I. Other desired Multiple analysis
F. Cross-spectral density analysis
G. Coherence analysis
H. Frequency response analysis

Table V—Analysis of Time Series in Rhythmometric Investigations


Time series
Classical "macroscopic" biometry of rhythms
"microscopic" rhythmometry
Classical biometry without quantification of biologic time structure
Complementary rhythmometry with quantification of biologic time structure
Histogram
Chronogram
Autocorrelation
Variance-spectrum
Mean
Median
Mode
Variance
Standard deviation
Standard error of mean
Confidence interval
Cross-correlation
Cross-spectrum
Coherence
Transfer-function
Covariance
Least squares spectrum
Correlation
Regression
Regressive amplitude and phase diagram
Amplitude-unweighted phase test
Phaseestimation from noise-to-signal ratio
Cosinor
Amplitude-weighted phase test

It turned out that this classification satisfactorily covered all biological methods. Still, it was necessary to make an additional subdivision within every class of methods depending on the level of analysis and to select accordingly: (1) the methods for detection of signals on a noisy background (D); (2) methods of phenomenological description (P); (3) analytical methods (A); and (4) methods of systemic analysis (S).

As an indicative method of qualitative analysis answering the question "what kind of", that is, to provide information on the general properties of an object or process, the following methods can be considered:

The methods of detection of signal in the presence of noise or separation of an object

As a result of applying these methods the following can be discovered.

—pure noise

—a signal or a complex of signals

The signal can be separated by the following means:

(1) Filtering—the detection of a signal with known parameters or elimination of noise with known frequency spectrum.

(2) Different ways of smoothing, like least squares method and special type filtering.

(3) Accumulation of signal (synchronous detection, correlation methods).

(4) Identification of signal presence by applying test stimuli.

Methods for testing of stationarity of the processes or, in general, testing for homogeneity within a group of objects

This method implies the reproducibility of characteristics of the process or object under given conditions and in some cases the ergodicity—the stationarity both in time and over the ensemble of realizations.

The following processes can be identified by comparing sequential fragments of the process or by sequentially measuring parameters of an object and also by comparing sequential fragments and separate realizations within the ensemble of realizations:

—nonstationary
—quasistationary
—stationary
—ergodic

Stationarity can be tested as follows:

(1) Calculation of the variance of the parameters as a measure of stationarity.
(2) Comparison of sequential fragments of the processes or sequentially measured parameters of an object, with the aid of statistical criteria and an analysis of the matrix of criterion values.
(3) Calculation of sequential or general correlation coefficients and their comparison in order to detect any dependence on the point of sampling.
(4) Plotting of the sequential spectra (Walsh, Fourier or other basis) to detect any dependence on the point of sampling.
(5) Calculation of the transition probability matrix and estimation of its numerical parameters.
(6) Calculation of reproducibility matrices—the matrices of a criterion value for the sequential fragments of the process.
(7) Selection of the optimal value of intervals of stationarity tested by different methods (primarily, variance analysis).

By applying these methods we learn whether we can use numerical parameters to describe the whole realization (as for stationary or ergodic processes) or only intervals of stationarity (in case of a quasistationary process).

In the absence of stationary fragments we must find other informative parameters, for instance the degree of stationarity.

Methods for analyzing the periodicity of the processes

As a result of their application we can determine whether the signal is:

—periodic with constant period
—quasiperiodic (with changing period)
—aperiodic

The detection of periodicity is possible with the help of the following methods:

(1) Methods of approximation.
(2) Correlation methods.
(3) Spectral methods.
(4) Method of periodograms.
(5) Analysis of extremal values.
(6) Cosinor method.
(7) The calculation of a matrix of reproducibility—the matrix of criterion values for sequential fragments of the process.

Methods for investigating the internal coherence of the process

As a result of this investigation the following can be obtained:

—Markov processes of the 1st, 2nd and higher orders.
—Coherent processes.

The test of coherence requires:

(1) Calculation of the transitional probability matrix and the estimation of the order of the Markov process.
(2) Estimation of the interval of coherence using the correlation function.
(3) Calculation of matrices of reproducibility.

In case a high degree of coherence is discovered filtering of data can be used to reduce their volume without the loss of information.

A high degree of coherence of a process also indicates high inertia of the studied object; that is of practical importance for the study of transient processes.

METHODS FOR EVALUATING NUMERICAL PARAMETERS OF OBJECTS, EVENTS AND PROCESSES

To answer the question "what kind of" in the narrow sense of the word are the following:

Different ways of estimating signal-to-noise ratio:

(1) Smoothing of numerical Data Sequences.
(2) Optimal linear and nonlinear filtering.
(3) Elimination of noise with known frequency parameters.

Methods for evaluating integral parameters of a process

They are different for periodic and aperiodic processes. For periodic processes the following methods are applicable:

(1) Analysis of the probability density distribution:
    (a) calculation of statistical moments (mean, variance, coefficients of asymmetry and

excess, and higher moments) and their comparison by means of parametric criteria.

(b) Calculation of distribution functions and their comparison by means of nonparametric criteria (rank correlation, Spearman, Mann-Witney, Kendall, Kolmogorov-Smirnov criteria and others).

(2) Calculation of reproducibility matrices.

(3) Evaluation of time series of values of the periods (spectrum of periods) by means of calculation of intervals between successive extremal values; crossing the zero line; and analysis of statistical properties of these time series.

(4) Calculation of the spectral density function using different bases.

(5) Investigation of the phase structure of processes analyzing the dependence between the accumulated phase shift and time.

(6) The "Cosinor" method.

(7) Calculation of the transfer function by means of operators method.

(8) Method of test stimuli.

(9) Plotting of the amplitude-frequency characteristics. For aperiodic processes we may apply:

(10) Analysis of amplitude distribution and calculation of moments.

(11) Analysis of power spectrum.

(12) Investigation of overshoots and their statistical properties.

(13) Calculation of the sequence of intervals between events.

(14) Investigation of patterns and their reproducibility.

Methods for detection of coherence of the process

Calculation of the transitional probability matrix and its numerical parameters (trace, determinant etc.).

ANALYSIS OF THE INTERCONNECTIONS BETWEEN ELEMENTS, PHENOMENA AND PROCESSES

As a separate parameter to be investigated (in response to the question "what kind of connection is this?"), we primarily selected the following features of interconnections:

a. Type of interconnection—linear or nonlinear. Within the nonlinear group we study those which can be represented by polynomials of higher order, harmonic functions, exponential, hyperbolic interconnections and so on.

b. Degree of linearity.

c. Direction of interconnection (vector of interconnection).

Tensor of interconnections can be used accordingly to describe completely the interconnections within the group of elements.

The first routine stage in the analysis of interconnection is the calculation of the correlation ratio indicating the presence or absence of any connection. If this parameter exceeds certain thresholds special tests concerning linearity of interconnection can be carried out. For that purpose the correlation coefficient reflecting the strength of linear (or approximately linear) interconnections is calculated.

The direction of interconnection can be obtained either by using the time shift of the principal maximum of the crosscorrelation function or by calculating the regression lines of each process relating to the other ones.

Nonlinear interconnections can be successfully approximated with linear functions within short intervals, principally for monotonic increasing or decreasing functions. In case such an approximation is not accurate enough and for periodical interconnections, we choose empirically approximating functions from routine methods (least squares, maximum likelihood, minimization of objective functions, nonlinear programming). The direction of interconnection can be specified by means of regression or factor analysis.

To investigate the interconnections on different structural levels and between the objects of different complexity we must take the following steps:

(1) In case the parameters of a single object are to be analyzed:

(a) Calculation of value and sign of correlation ratio.

(b) Calculation of correlation coefficient to estimate the degree of linearity of the interconnection.

(2) To analyze the interconnections between elements and elementary events, calculate vector of interconnections.

(3) When analyzing the interconnections within the system calculate tensor of interconnections.

METHODS OF INVESTIGATING THE EXTERNAL PROPERTIES OF PROCESSES AND OBJECTS

These are relating to the environment and answering the questions "what kind of," "when," "where" and "what".

Methods of calculating the unobservable

Observation relates here to the field of metrology space-time coordinates. Such calculations are carried out with the aid of:

(1) Crosscorrelation analysis.

(2) Description of objects' motion in space-time coordinates by means of differential equations.
(3) Calculation of dynamic transfer functions (related to relaxation processes) by routine methods.
(4) Evaluation of accumulated phase shift as a function of time.

Methods of sorting and distributing objects according to certain indicators

(1) Calculation of statistical moments.
(2) Calculation of distribution function.
(3) Calculation of the multidimensional joint probability density functions and estimation of their parameters.

Methods of analysis of motion and functioning of an object in a space-time coordinate system

(1) Calculation of matrices of reproducibility.
(2) Construction and solution of pertinent differential equations.

Methods of identifying and classifying objects, events and processes

As a possible way to answer the question "what" we determine whether one observed phenomenon is similar to another. The problem is to identify an object as belonging to a certain class—it is pattern recognition. In such cases the following methods could be applied:

(1) Search for the complete or partial precedent.
(2) Diagnostic procedures in multidimensional space on the basis of Bayesian or other criteria.
(3) Calculation of discriminant functions of the first and higher orders.

Cases where there is no a priori information on classes

Their images can be constructed using:

(1) Factor analysis—principal components in particular.
(2) Cluster-analysis.
(3) Adaptive classification (with learning).

The enumeration of these methods shows that any one method can belong to more than one group (like correlation analysis which can be applied to signal detection, to testing the periodicity and for estimating the numerical parameters of the process). From our point of view, this does not make the suggested classification less convenient because of the different interpretations of the results in each case.

The complete classification is given in Table VI where the arrows denote what questions can be answered by means of a particular method. Of course, not all the possible mathematical methods are exhibited but only those widely used in the solution of biomedical processing tasks according to the literature.

Not being mathematical specialists we have undoubtedly missed a few "tools" which may appear more or less important to others, but the open-ended structure of the classification scheme allows us to add anything necessary. There is also no doubt that from the pure mathematician's point of view this classification is rather eclectic. However, our aim was not mathematical harmony but the distribution of analytical tools over the "shelves" to make it convenient for the user to take the needed one from there according to his/her requirements.

FEASIBLE FORMULATION OF BIOMEDICAL PROCESSING TASKS

The difficulty of choosing an adequate method of analysis derives not only from the above mentioned large number of possible methods and the complexity of the problems themselves. This was only the tip of the iceberg. Complications arise when optimal time periods must be selected (also the correct digitizing interval) for a given method in order to find the desired parameters. However, it is impossible to discuss these problems here in detail.

The way to overcome these difficulties seems to be formalization of the approach to the statement of biomedical data processing tasks, thus guaranteeing the correct approach.

As a first step along this line we suggest the use of a standard form with the following items:

(1) Parameter (or parameters) to be investigated.
(2) Intervals of parameter values and the given precision of the measurement of each one.
(3) Duration of observations (adding criteria of choice if there are any).
(4) Digitizing interval (number of samples per observation).
(5) Object of analysis (elements, elementary events, processes, interconnections).
(6) The real aim of the analysis (to answer the questions "What kind of," "Where," "When," "What").
(7) The final aim of the analysis (biomedical problem).

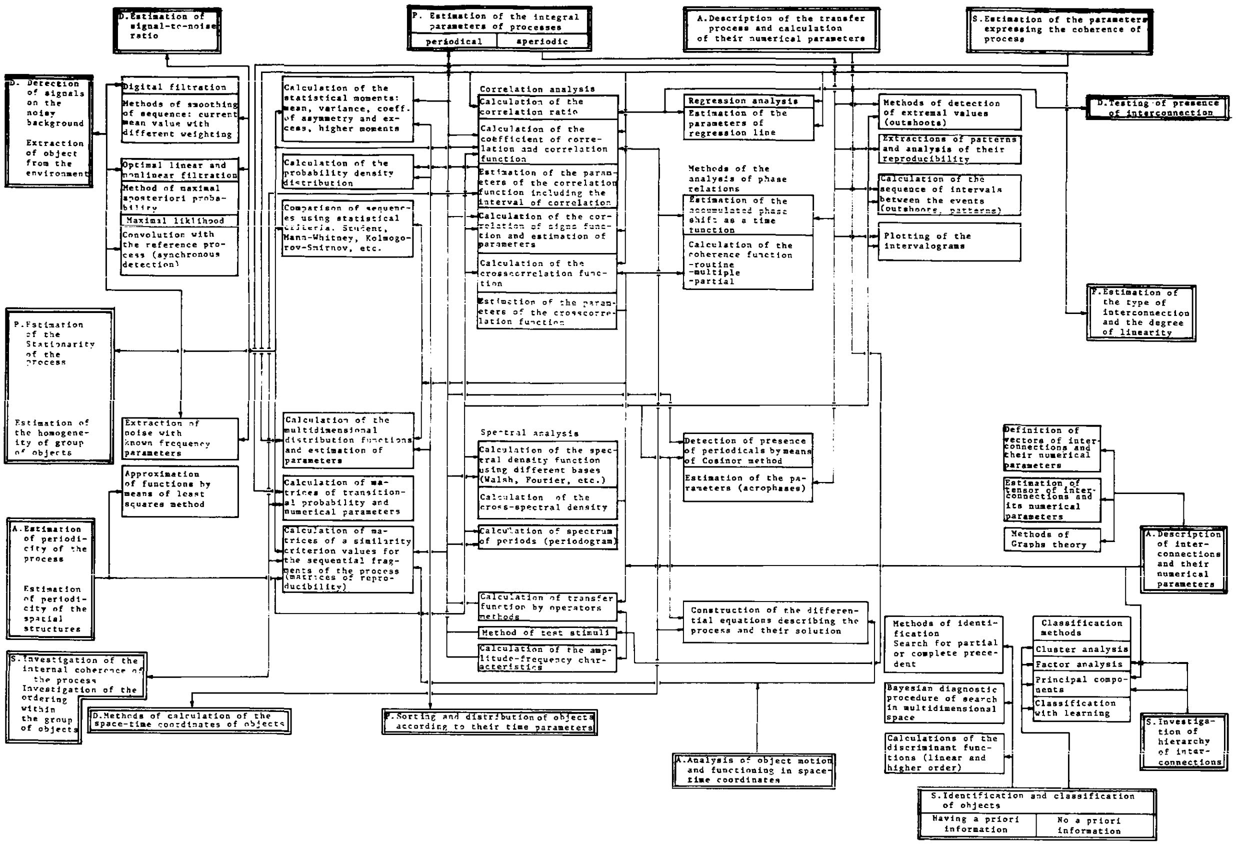
These seven pieces of information are supplied by the research worker; the people who deal with the data processing tasks should add one more:

(8) Results of preliminary analysis of data by means of indicative methods (general information about the properties of the object or process).

Here the following are useful:

(a) signal-to-noise ratio.
(b) whether the process is stationary and the in-

TABLE VI—Quantitative Methods of Investigation of Internal Properties of Objects, Events and Processes


D.Estimation of signal-to-noise ratio
P. Estimation of the integral parameters of processes
periodical
aperiodic
A.Description of the transfer process and calculation of their numerical parameters
S.Estimation of the parameters expressing the coherence of process
D. Detection of signals on the noisy background
Extraction of object from the environment
Digital filtration
Methods of smoothing of sequence: current mean value with different weighting
Optimal linear and nonlinear filtration
Method of maximal aposteriori probability
Maximal liklihood
Convolution with the reference process (synchronous detection)
Calculation of the statistical moments: mean, variance, coeff. of asymmetry and excess, higher moments
Calculation of the probability density distribution
Comparison of sequences using statistical criteria. Student, Mann-Whitney, Kolmogorov-Smirnov, etc.
Correlation analysis
Calculation of the correlation ratio
Calculation of the coefficient of correlation and correlation function
Estimation of the parameters of the correlation function including the interval of correlation
Calculation of the correlation of signs function and estimation of parameters
Calculation of the crosscorrelation function
Estimation of the parameters of the crosscorrelation function
Regression analysis
Estimation of the parameters of regression line
Methods of the analysis of phase relations
Estimation of the accumulated phase shift as a time function
Calculation of the coherence function
-routine
-multiple
-partial
Methods of detection of extremal values (outshoots)
Extractions of patterns and analysis of their reproducibility
Calculation of the sequence of intervals between the events (outshoots, patterns)
Plotting of the intervalograms
D.Testing of presence of interconnection
F.Estimation of the type of interconnection and the degree of linearity
P.Estimation of the Stationarity of the Process
Estimation of the homogeneity of group of objects
Extraction of noise with known frequency parameters
Approximation of functions by means of least squares method
Calculation of the multidimensional distribution functions and estimation of parameters
Calculation of matrices of transitional probability and numerical parameters
Calculation of matrices of a similarity criterion values for the sequential fragments of the process (matrices of reproducibility)
Spectral analysis
Calculation of the spectral density function using different bases (Walsh, Fourier, etc.)
Calculation of the cross-spectral density
Calculation of spectrum of periods (periodogram)
Calculation of transfer function by operators methods
Method of test stimuli
Calculation of the amplitude-frequency characteristics
Detection of presence of periodicals by means of Cosinor method
Estimation of the parameters (acrophases)
Definition of vectors of interconnections and their numerical parameters
Estimation of tensor of interconnections and its numerical parameters
Methods of Graphs theory
A.Description of interconnections and their numerical parameters
A.Estimation of periodicity of the process
Estimation of periodicity of the spatial structures
S.Investigation of the internal coherence of the process
Investigation of the ordering within the group of objects
D.Methods of calculation of the space-time coordinates of objects
P.Sorting and distribution of objects according to their time parameters
Construction of the differential equations describing the process and their solution
Methods of identification
Search for partial or complete precedent
Bayesian diagnostic procedure of search in multidimensional space
Calculations of the discriminant functions (linear and higher order)
Classification methods
Cluster analysis
Factor analysis
Principal components
Classification with learning
S.Investigation of hierarchy of interconnections
A.Analysis of object motion and functioning in space-time coordinates
S.Identification and classification of objects
Having a priori information
No a priori information

terval of stationarity.

(c) statistical parameters, primarily the type of distribution.

(d) whether the process is periodical.

(e) results of Markov property test.

Analysis of data thus obtained in combination with the suggested classification schema will facilitate the choice or the optimal method of data processing. In case there are several possible methods one should remember the Okkam razor and choose the simplest one so as not to increase unnecessarily the number of essential descriptors.

In conclusion we would like to stress that the choice of data processing methods must be based upon their applicability to the data at hand. A bad choice will lead to the data having to be "reprepared" in a perhaps less useful format.

REFERENCES

1. Moiseeva, N. I. and M. Yu. Simonov, "Some Limitations of Computer Data Processing," *Problems of Cybernetics*, Vol. 24 and *Problems of Medical Cybernetics*, Moscow, 1975, pp. 98-112.
2. Petrov, M. I., "Systemic Aspects of Scientific Activity," *Systemic Investigations*, Moscow, 1972, p. 34.
3. Schrōdinger, E., *What is Life? The Physical Aspect of the Living Cell*, 1955.
4. Moiseeva, N. I., "Medical Aspects of Computer Diagnostics in Neurology," *Meditsina*, Leningrad, 1972.
5. Gontcharov, V. A., "On the Problem of Tasks Classification," *Controlling Systems and Machines*, 1975, N. 1, pp. 95-97.
6. Dixon, W. J., "Statistical Packages in Biomedical Computation," *Computers in Biomedical Research*, Vol. I, Eds: R. W. Stacy, and B. Wasman, New York, London, Academic Press, 1965, pp. 47-64.
7. Rashevsky, N., *Some Medical Aspects of Mathematical Biology*, Ch. C. Thomas, Springfield, Illinois, 1964.
8. Urbach, V. Yu., *Mathematical Statistics for Biologists and Physicians*, Moscow, Acad. Sci. Press, 1963.
9. Urbach, V. Yu., "Biometrical Methods," Moscow, *Nauka*, 1964.
10. Urbach, V. Yu., "Statistical Analysis in Biomedical Research," *Meditsina*, Moscow, 1975.
11. Ivanov-Muromsky, K. A. and S. Ja. Zaslavsky, "Application of the Computer for Brain Electrodramms Analysis," *Naukova dumka*, Kiew, 1968.
12. Moiseeva, N. I. and V. V. Usov, "Some Mathematical Aspects of Computer Diagnosis," *Proceedings of the IEEE*, Vol. 57, N. 11, 1969, pp. 1919-1925.
13. Heinmets, F. (Ed.), "Concepts and Models of Biomathematics," *Biomathematics*, A series of Monographs, New York, 1969.
14. Plochinsky, N. A., *Biometry*, Moscow University Press, 1970.

15. Gorovenko, G. G., V. A. Diadura, N. N. Zukov, M. M. Petrov, B. V. Radionov and E. L. Tsatsko, "Probability and Statistical Methods and Computers in Biomedical Research," *Zdorovie*, Kiew, 1970

16. Kline, N. and E. Laska, *Computers and Electronic Devices in Psychiatry*, Grune and Stratton, New York, London, 1968.

17. Reinberg, A., "Methodological Considerations for Human Chronobiology," *Journal of Interdiscipl. Cycle Res.*, Vol. 2, N. 1, 1971, pp. 1-15.

18. Lakin, G. F., "Biometry," *Vysshaya Skola*, Moscow, 1973.

19. Bendat, J. S. and A. G. Piersol, *Random Data: Analysis and Measurement Procedures*, New York, London, Wiley Interscience, 1970.

20. Hallberg, F., "Chronobiology," *Annual Review of Physiology*, Vol. 31, 1969, pp. 675-725.

An adaptable, modular data-collection system suitable for scientific experimentation — Analog to digital transformation, short-term digital storage, formatted digital tape-recording, and computer entry of experimental data

by HAROLD H. SHLEVIN
University of Rochester School of Medicine
Rochester, New York

ABSTRACT *

This paper describes an adaptable, modular data acquisition system suitable for the collection of experimental data in a form readily adapted to subsequent digital computer entry and analysis. The system utilizes voltage-to-frequency conversion of an input analog signal over precisely regulated timing intervals. The digitized data is initially stored in shift-registers and later digitally formatted and digitally tape-recorded. The recorded data is read by a complimentary tape reading system and entered into a mini-computer for subsequent analysis and long-term storage. The digital formatting technique permits unambiguous definition and recovery of each data word as the tape is read.

Adaptations of this system necessary to accommodate variations in the length of each data-word and the number of words per message block are discussed.

Commercially available TTL integrated circuits and modular components are used in the design of this system.

INTRODUCTION

Scientific experiments often require a means of rapidly collecting and storing analog experimental data in a form readily adapted to later digital computer entry and analysis Although commercial systems are available, individual systems better suited for the particular experimental configuration can be constructed with relative ease and at significantly reduced cost by using readily available TTL and CMOS integrated-circuits.

The data collection system described in this paper evolved from a desire to carry out electrophysiological experiments aimed at further characterizing the membrane charge movement phenomenons in single, skeletal muscle cells (Schneider, M. F. and Chandler, W. K., 1973). A Digital Equipment Corporation PDP-8/E minicomputer was available for off-line analysis. Therefore, an interim means of storing the data for later computer entry was needed.

This paper describes an adaptable, modular data-collection system suitable for the collection of experimental data. The description is divided into four logical segments (Figure 1): (1) Data transformation using voltage-to-frequency conversion, and static shift-register storage of the digitized data, (2) Formatting of the stored, digital data using a Universal Asynchronous Receiver-Transmitter (U.A.R.T.), in preparation for tape-recording, (3) Digital tape-recording, and, (4) Reading and computer entry of the digitally tape-recorded data.

DATA TRANSFORMATION

The process of analog-to-digital conversion of the experimental data was simplified by using a voltage-to-

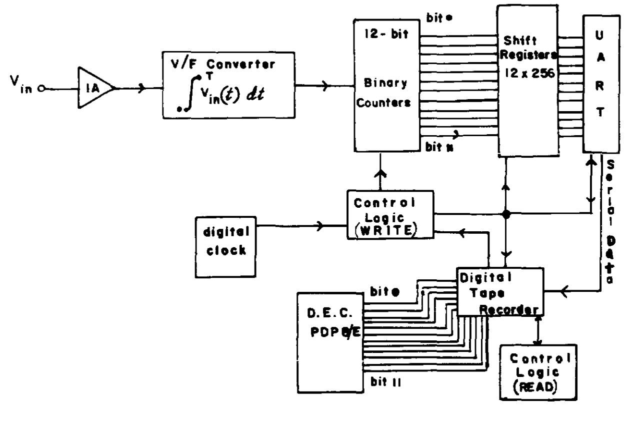


Figure 1—Block diagram of the data-collection system

* This work has been supported by NIH/USPHS Training Grant number 5-T01-GM00394; and in part by an M.D.A.A. research grant to Dr. Martin F. Schneider.

frequency (V/F) converter module (Anadex Instruments, Inc. Model 1700-5044-00). The converter produces TTL/DTL compatible output frequencies of 0-1.0 MHz that are directly proportional to input D.C. voltage signals of 0-10 V.D.C. The linearity of the converter is better than 0.01 percent of full-scale, and its output responds virtually instantaneously to changes in input voltage. Experimental input D.C. voltages of ±100 mV. in amplitude are amplified to a level commensurate with that of the V/F converter by an instrumentation amplifier (Analog Devices, Inc. Model AD520J) with externally adjustable gains of 1, 20, 50, or 100, and adjustable reference levels.

Voltage-to-frequency converters require less complex wiring for data transmission than do conventional analog-to-digital converters; only a single line is needed for a serial pulse-train. A clock signal and a counter at the receiving end are used to encode the pulse train. The process results in an analog-to-digital conversion and time-integration of the input signal. This was particularly desirable for later data analysis (Figure 1).

The output pulses of the V/F converter were counted by three cascaded four-bit binary counters (SN 74193) for externally adjusted intervals of 0.1, 0.5, 1.0, 2.0, 5.0, or 10.0 milli-seconds. Timing pulses, which had been buffered to TTL levels, were generated by a gated, pulse train output channel of a digital stimulator (Digitimer Ltd., Hertfordshire, England, Model 4030). When the binary counters for the V/F converter output receive a timing signal, the following sequence of events occur (Figure 2): (1) The input to the counters from the V/F converter is disabled, (2) The twelve-bit binary number presently in the counters is transferred to a digital (12 X 256)—bit storage register (NS 5055), (3) The binary counters are cleared, (4) A Master-Event-Counter is advanced by one count, and (5) The input to the V/F counters is again enabled. These five steps occur within 3.8 micro-seconds. By repetition of this sequence, a total of 256 sequential data, points are sampled during any collection interval, (e.g. one oscilloscope sweep).

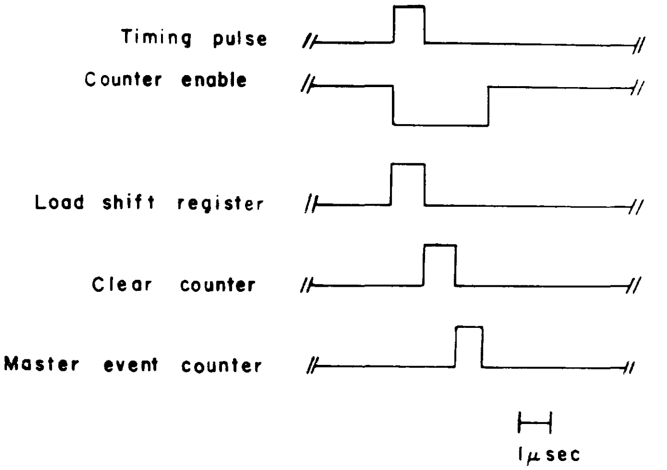


Figure 2—Voltage-to-frequency converter; Timing sequence

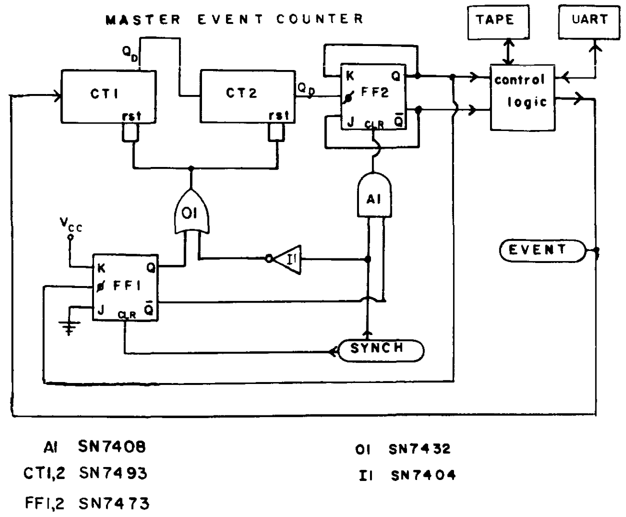


Figure 3—Master-event-counter; Circuit diagram arrows indicate direction of information flow
(A1=SN 7408; CT1, CT2=SN 7493; FF1, FF2=SN 7473; 01=SN 7432; and I1=SN 7404)

A Master-Event-Counter circuit serves to control two distinct sequences (Figures 1, 3, and 4): (1) The actual digital transformation of the analog data and the static storage of 256 digital data points, and (2) The digital formatting and then digital tape-recording of the previously collected 256, twelve-bit binary data words.

In conjunction with an initializing SYNCH pulse, CT1 and CT2 (cascaded four-bit binary counters SN 7493), and FF1 and FF2 (dual J-K flip flops with clear SN7473) are cleared. This sets $\overline{QFF2}$ to logic high and enables the Master-Event-Counter and the sequence of events necessary to digitally transform and store the experimental data signal, as previously described. After 256 data points have been sampled, QFF2 toggles to logic high enabling the sequence of events required to format the 256 twelve-bit binary data words, and to initialize and consummate the

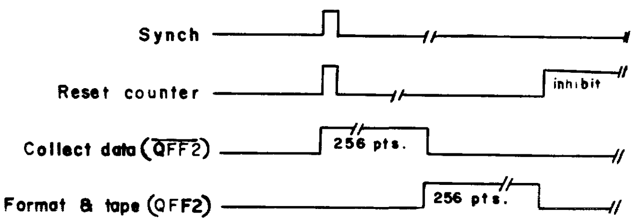


Figure 4—Master-event-counter; Timing sequence

taping process. After taping of all 256 data words has been completed, FF2 again toggles setting QFF1 to logic high. This blocks any further timing pulses and blocks all counters until another SYNCH pulse is received.

Although independent and variable clocks are used to collect the data and to format and tape the data, the control logic segment assures that only a single Master-Event-Counter pulse is generated for each twelve-bit binary data word collected and for each original twelve-bit data word transmitted by the U.A.R.T. to the tape-recorder.

Although the data-collection system is designed specifically for the digital transformation and taping of 256 twelve-bit binary data words, it is easily adapted to any multiple of 2^n points by simply changing the Master-Event-Counter, the length and/or width of the shift-register buffer, and the number of bits per data word as may be appropriate for the resolution desired.

FORMATTING OF THE STORED DATA WORDS

A Universal - Asynchronous - Receiver/Transmitter (U.A.R.T.) in half-duplex mode was used to ensure the competency of the data interface between the 256 word shift-register buffer and transmission to the digital tape-recorder The transmitter section of the U.A.R.T. (Texas Instrument Co. Model TMS 6011NC) accepts the parallel data from the shift-registers, converts it to serial form and generates a start, parity, and stop (1 to 2) bit(s) for each original twelve-bit binary data word. The data word length (5 to 8 bits long), the baud rate (0 to 200 KHz.), and the sign of a parity bit are externally selected. The formatting function and parity generation of the U.A.R.T. allow for unambiguous definition of each data word when reading the data tape and for the detection of dropped bits.

To transmit one twelve-bit word from the shift-registers to the tape-recorder, two ten-bit data words are sent by the U.A.R.T. Each word is formatted as shown in Figure 5. Each half (six bits) of the original twelve-bit data word is concatenated to an indicator bit. The indicator bit is used to indicate whether we have the front-half or the back-half of the original twelve-bit data word; this is rechecked when reading the tape.

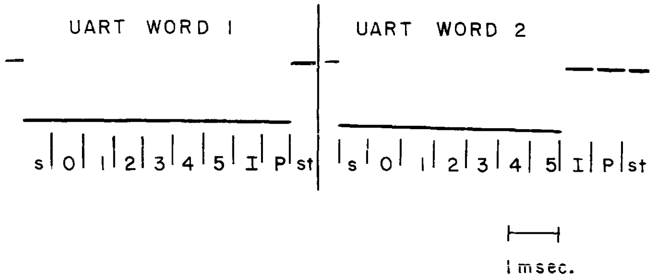


Figure 5(b)—Actual oscillographic record of formatted output from the U.A.R.T. sent to WRITE unit

The U.A.R.T. clock, derived from an internal oscillator, runs at a frequency of sixteen times the baud rate. Successive divisions of the U.A.R.T. clock rate by sixteen (for the baud rate), by ten (signalling each half-word), and then by two (denoting complete transmission of the two halves of the original twelve-bit data word), together with appropriate internal and external control logic allow for constant checking and control over the formatting and transmission processes. For each original twelve-bit data word, transmitted as two ten-bit, formatted U.A.R.T. words, the Master-Event-Counter is advanced by one count.

This configuration is easily adapted to any length data word by appropriate fragmentation of the data word into separate U.A.R.T. words, indicators bit(s), and appropriate division of the U.A.R.T. clock to form the clock pulse which serves as input to the Master-Event-Counter.

DIGITAL TAPE RECORDING

The core of the system is a MicroVox tape recording/reading unit (Micro Communications Corporation, Waltham, Massachusetts). There are two separate, hand-size units in this system: a WRITE unit and a READ unit. Each unit consists of a tape drive system, plus all the necessary data transfer and tape handling

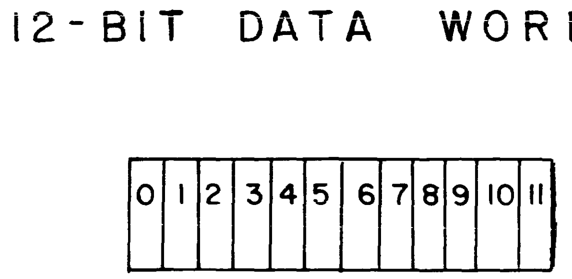


Figure 5—Formatting. (a)—Original twelve bit binary word from counters for V/F converter

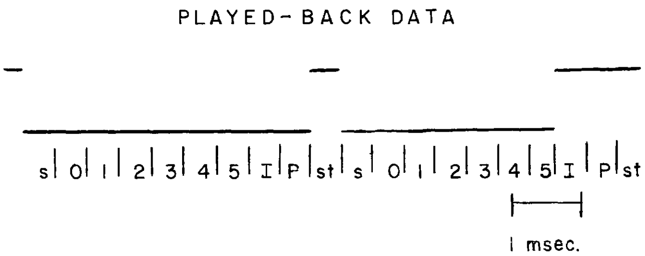


Figure 5(c)—Actual oscillographic record of DATA recovered from the READ unit, on expanded time base for comparison with Figure 5b logic. The available lines are: CLOCK_IN, CLOCK_OUT, DATA, READY, END_OF_TAPE/BEGINNING_OF_TAPE, ON, and FAST_FORWARD. The internal CMOS circuitry of the MicroVox units provides true TTL-compatible logic interfaces (after appropriate buffering) to and from the data-collection system.

The WRITE unit is functionally a phase-lock loop (P.L.L.) (Figure 6). The P.L.L. functions as an electronic servo consisting of a phase detector, a low pass filter, and a voltage-controlled oscillator (V.C.O.). The V.C.O. enables the P.L.L. to synchronize or "lock" on the incoming clock signal. The clock signal (CLKIN) serves to clock in each bit of the word. As the phase of the clock signal changes, indicating a change in incoming frequency, the output of the phase detector will change just enough to keep the V.C.O. frequency the same as the incoming frequency. This allows the baud rate to be varied both dynamically and statically over a wide range.

In external-clock mode, the MicroVox WRITE system's P.L.L. synchronizes with the leading edge of a user supplied clock-in (CLKIN) signal. This signal is derived from the successive division of the U.A.R.T. clock signal (÷ 16), described earlier. The WRITE system furnishes an externally available clock-out (CLKOUT) signal which is checked by the data-collection system to verify phase-locking quality. A modulator, internal to the WRITE system, then manufactures the waveforms of logic 1 and logic 0 which are written onto the tape in proper relation and synchrony with the formatted, serial data stream from the U.A.R.T. The MicroVox WRITE system can function over a range of 300 to 3200 bits/second.

This technique of pulse-coded modulation affords significant advantages over more conventional AM or FM recording methods: (1) A greater accuracy, (2) A significantly better signal-to-noise ratio (up to 90 db), (3) The capability for relatively long-distance transmission of the data without degradation since the digital signal can be regenerated with a very low probability of loss, and (4) The ability to computer process the information with virtually no modifications.

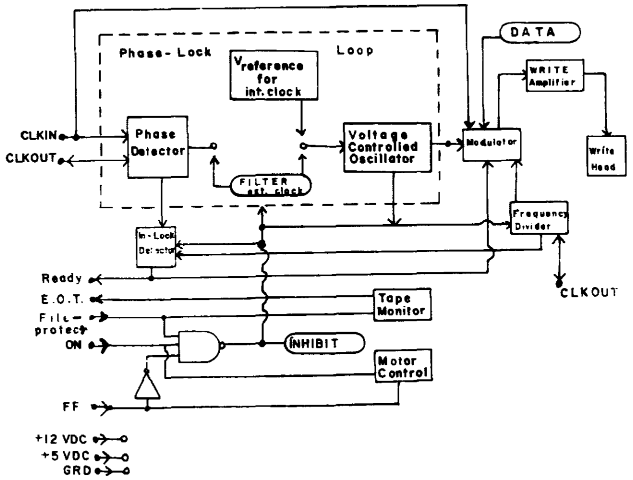


Figure 6—MicroVox WRITE unit: Functional block diagram reproduced with permission of Micro Communications Corporation

The tapes are continuous loop, one-track cartridges available in a variety of lengths. Since the tape is a loop, the physical end-of-tape (EOT) and beginning-of-tape (BOT) are the same. Provisions exist for detection of EOT/BOT as well as for file protecting each cartridge.

Transition of QFF2 of the Master-Event-Counter from logic low to logic high initializes the following sequence of events by the data-collection system leading to consummation of the WRITE function (Figures 6, 7): (1) Fast_Forward (FF) is set to logic 0 and ON to logic 1, thereby entering WRITE mode and accelerating the tape to speed. If initially stopped, the tape unit reaches speed within 60 milli-seconds. (2) The CLKIN signal derived from successive division of the U.A.R.T. clock is supplied. The MicroVox WRITE unit then furnishes an external READY (RDY) signal indicating that phase-lock and speed have been attained. This permits the external data-collection interface to transmit DATA to the recorder and allows the recorder to write the formatted DATA onto the tape. Had CLKIN not been supplied, the RDY signal would not be furnished and the data-collection interface would not permit the sending of DATA to the recorder. During such a state, the MicroVox WRITE unit would write bias onto the tape; this prevents mangling of message blocks and later allows the MicroVox READ system to start and stop between two consecutive message blocks. (3) Each data bit is clocked to the recorder by the CLKIN signal. There is a delay of approximately 1½ CLKIN cycles between the occurrence of a CLKIN pulse and the actual writing of that particular data bit onto the tape.

In this fashion all 256 original twelve-bit data words (or 5120 U.A.R.T. bits) are transmitted. Upon com-

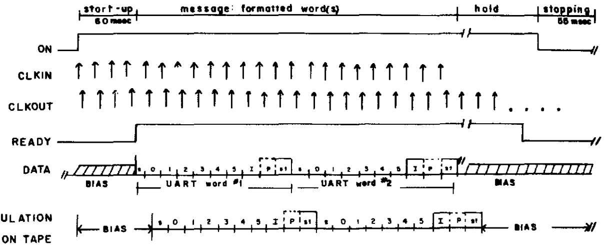


Figure 7—MicroVox WRITE system: Operating sequence A formatted U.A.R.T. word is shown: s=start bit, 0-5 denote data bits, I=indicator bit (0 or 1), P=parity (odd or even), and st=stop bit (1 shown)

pletion of transmission, QFF2 of the Master-Event-Counter flips to logic low and in conjunction with a U.A.R.T. done flag, the CLKIN signal is discontinued. After 1½ bit periods, the time needed to write the last bit, RDY drops to logic 0. (4) The data-collection interface then sets ON to logic 0 and the tape dynamically brakes to a standstill.

The external data-collection interface provides the option of slewing to the EOT/BOT in Fast_Forward or slow mode (with CLKIN discontinued, FF=1, ON=1). EOT/BOT is optically sensed by the MicroVox WRITE and READ systems from two foil-reflectors spaced 3″ apart and affixed to the tape.

During the writing of data, the occurrence of the first EOT applique causes the data-collection interface to finish writing only the current twelve-bit word (20 U.A.R.T. bits), and then the recorder is stopped. A new cartridge is inserted, and writing resumes under control of the Master-Event-Counter. However, in slew to EOT/BOT mode, the external interface stops the tape immediately after the second EOT applique in preparation for later reading or writing.

No modifications to either the data-collection interface controlling the MicroVox WRITE system (other than those already described) or to the tape unit itself are necessary to write any word-length, or number of words per data block.

DIGITAL TAPE READING AND COMPUTER ENTRY

The MicroVox READ system is specifically designed to read tapes produced by the MicroVox WRITE system. A high degree of cooperativity between the two units minimizes errors that could result from tape speed irregularities (e.g., wow and flutter), allows for writing and reading the same tape-wafer at widely different speeds, and results in simple adaptation of the tape units to a large number of external interfaces.

The instrumentation of the motor control functions (ON, FF, EOT/BOT) of the READ system are identical to those of the WRITE system; their functions are similarly controlled by the external data-collection interface. The bit density (b.p.i.) written on the tape, and the speed (i.p.s.) at which the tape is read determine the waveform presented to the READ head. Above a density of 800 b.p.i., the READ waveform is no longer usable. A dual-peak-detecting system, with a decent frequency response and insensitive to both amplitude and waveform, is used to allow for large variations in the amplitude, frequency, and shape of the waveform presented to the READ head.

To read a tape-wafer, the external data-collection interface is used to control the following sequence of events (Figure 8): (1) Fast Forward is set to logic 0 and ON to logic 1, thereby entering READ mode and accelerating the tape to speed. (2) The READY signal rises to logic 1 a full bit-interval before the first data bit is shifted to the external interface, allowing the maximum possible time in which to ready the external data interface. (3) A CLKOUT pulse is furnished by the READ unit approximately 5 μsecs. after each DATA bit is shifted. The process continues until modulation ceases, then (4) READY drops to logic low, and depending upon the states of the external interface and the computer program either the tape is stopped (ON to logic 0) or reading is continued (ON remains at logic 1).

Data collection system—computer interface

The use of the U.A.R.T. to format the data sent to the MicroVox WRITE system allows the data collection interface to unambiguously define the beginning, end, and origin (front-half or back-half of the original twelve-bit data word) of the words being read.

The start of each U.A.R.T.—word is defined as a transition of the READ system bit—output from logic high to logic low. This transition is detected by the external data-collection interface which is preset for the word-length, the parity (even or odd) and the number of stop bits (1 or 2) written.

Detection of a start-bit by the external interface enables the following sequence of events (Figure 8): (1) A word-length counter (SN 7490), parity generator (SN 74180), and a ten-bit series-to-parallel shift-register (SN 7496) are cleared. (2) Each bit of the ten-bit U.A.R.T.-word is clocked into the ten-bit series-to-parallel shift-register by the CLKOUT signal derived from the MicroVox READ unit. When a complete ten-bit U.A.R.T.-word has been shifted, (3) The word-length counter overflows, (4) Parity is checked and error flag(s) set as necessary, (5) Either the first-six or last-six bits of a twelve-bit parallel-in, parallel-out

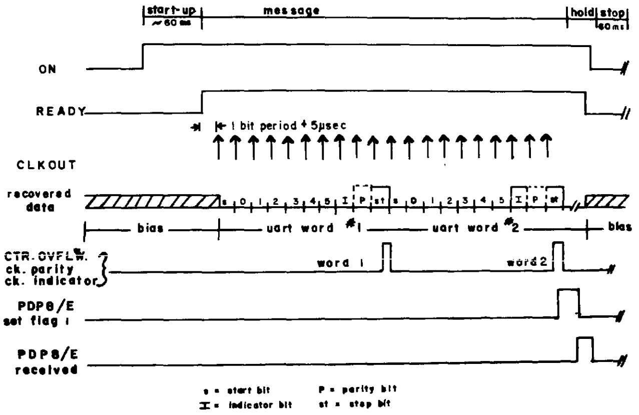


Figure 8—MicroVox READ system and DATA interface operation

A recovered DATA word is shown: s=start bit, 0-5 denote data bits, I=indicator bit (0 or 1), P=parity (odd or even), and st=stop bit (1 shown) buffer register (SN 74174) are loaded (with the data bits of the U.A.R.T. word) depending on whether the indicator-bit is a logic 1 or a logic 0, respectively. These five steps are repeated until the second U.A.R.T.-word has been received; then, (6) The data-collection interface flags the computer (D.E.C.—PDP-8/E) and a complete twelve-bit binary data word is read into core memory. (7) In response, the computer acknowledges receipt of the data word.

This process is continued until a total of 256 twelve-bit, binary data words (1 sweep) have been received and acknowledged by the computer. The controlling program then either stops the READ unit or continues the reading process as may be appropriate.

A Digital Equipment Corporation PDP-8/E mini-computer was used to control the read portion of the data collection system, and perform subsequent data analysis. During the READ cycle the data collection system communicates with the PDP-8/E through a D.E.C. M1709 omnibus interface module.

The computer was programmed in OS/8 Fortran IV. The actual control of the data-collection system is implemented by a RALF assembly language subroutine. The starting and stopping of the READ unit, the number of 256-word data blocks read, the handling of all error flags, and the transfer of the core resident data to Dectape for long term storage are all under program control. To maximize efficient use of storage, three twelve-bit binary data words are stored in both core and on Dectape in the space allocated for one Fortran floating point variable.

In subsequent analysis of such condensed data it is necessary to "float" each twelve-bit, binary data word into one Fortran floating-point variable. Each Fortran floating-point variable occupies 3 contiguous storage locations. This is accomplished by executing a call from the main FTN IV program to a specific RALF subroutine which floats a binary word. All program generated output destined for Dectape is recondensed before transfer.

The programs and interfaces are easily adapted to accommodate a variety of word-lengths, words per message block and other necessary control functions.

ACKNOWLEDGMENTS

It is a pleasure to thank Dr. Martin F. Schneider for his helpful advice and discussion concerning this system, as well as his continual encouragement.

I am grateful to my wife, Barbara for her continual encouragement and love, and for her help with the illustrations; and to Mrs. Lillian Peracchia for help with photography.

I thank Mr. John Young of the Department of Biomathematics for his helpful discussion and debugging aide; and also thank Irma Johnson, and John Hemmert of Micro Communications Corporation for much helpful advice.

This work has been supported by NIH/USPHS Training Grant #5-T01-GM00394 and in part by a research grant from the Muscular Dystrophy Associations of America to Dr. Martin F. Schneider.

The author gratefully acknowledges the support of Micro Communications Corporation for travel-expenses necessary to attend this conference.

REFERENCES

1. Micro Communications Corporation, Equipment Manuals: *MicroVox Digital Write System, and MicroVox Digital Read System,* with permission, 1975.
2. Schneider, M. F. and W. K. Chandler, 1973, "Voltage-Dependent Charge Movement in Skeletal Muscle; A Possible Step in Excitation Contraction Coupling," *Nature,* London 242, pp. 244-246.

Classification of personal information for privacy protection purposes*

by REIN TURN
The Rand Corporation
Santa Monica, California

ABSTRACT

Laws now in effect require protection of individual privacy in personal information record-keeping systems maintained by the federal government and by several states. These requirements will be extended to the private sphere in the future. It is necessary for their implementation in record-keeping systems to establish a standard sensitivity scale and classification system for personal information. This paper surveys several classification systems that have been discussed in the literature, examines the criteria for setting up such systems, proposes a new sensitivity scale and corresponding classification system, discusses the information integrity and security provisions that should be adequate for each classification level, and examines the problems that arise in assigning sensitivity and classification levels to personal information items, records, and record-keeping systems.

INTRODUCTION

The federal Privacy Act of 1974[1] and similar laws in several states (Minnesota, Arkansas and Utah) have established certain rights of individuals regarding personal information maintained on them by government agencies, restricted the use and dissemination of personal information, and prescribed requirements for information quality, integrity and security. In particular, the Privacy Act of 1974 states that any agency of the federal government must "maintain all records which are used by the agency in making determinations about an individual with such accuracy, relevance, timeliness and completeness as is reasonably necessary to assure fairness to the individual in the determination," and must "establish appropriate administrative, technical, and physical safeguards to insure the security and confidentiality of records and to protect against any anticipated threats or hazards to their security or integrity which could result in substantial harm, embarrassment, inconveniences or unfairness to any individual on whom information is maintained." The Act and state privacy laws also note that certain personal information items are available to anyone under the provisions of the Freedom of Information Act,[2] certain other items must be restricted to the agency personnel on a need-to-know basis, and still other items may be withheld even from the individual data subject himself.

The privacy protection laws recognize implicitly that not all items of personal information are equally critical in making a fair determination about an individual, that they are not equally sensitive from the point of view of their dissemination causing harm or embarrassment to an individual, and that the information quality, integrity and security requirements may vary among information items, types of records, and types of record-keeping systems. Even the same information item may be innocuous in one system of records, but very sensitive in another. For example, while a person's name is usually public information, it becomes sensitive when associated with a system of psychiatric treatment records.

Since most of the personal information record-keeping systems do not contain highly sensitive information, it is not necessary nor would it be economically practical to require absolute quality, integrity and security for all personal information in all record-keeping systems. Rather, following the approach taken in handling sensitive national defense information, a set of information sensitivity categories could be established such that, for each category, the access and dissemination restrictions would be specified and the minimal levels of required information quality, integrity and security would be defined. Thus, the technical questions of assuring quality and integrity, and providing security would be separated from the social policy questions of determining what level of integrity and protection must be provided for a particular type of information in a particular record-keeping system—several sensitivity categories are available and the

* Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views of The Rand Corporation or the official opinion or policy of any of its governmental or private research sponsors.

corresponding access control, integrity and security levels are provided by the system, such that any information item can be assigned to the sensitivity category most suitable under the circumstances.

This paper surveys several sensitivity classification systems that have been discussed in the literature, examines the criteria for setting up such systems, and proposes a generalized set of sensitivity categories for personal information in governmental as well as private record-keeping systems.

PROPOSED CLASSIFICATION SYSTEMS

Several suggestions for classification of personal information have been made in the literature. One of the earliest proposals by Comber[3] defines three categories based on dissemination controls that are applied:

1. *Unclassified*—All data maintained by a public agency not otherwise classified as restricted or confidential.
2. *Restricted*—Data that are not prohibited from full and free disclosure by statute (confidential), but whose unauthorized use could constitute an unwarranted invasion of personal privacy.
3. *Confidential*—Data that are prohibited from free and full disclosure by statutory regulation (law).

Comber suggests that among the criteria for classifying personal information as "restricted" should be whether or not the disclosure of data in question would: (1) Facilitate unwarranted identification of individuals, (2) Cause unjust economic loss or public stigma or harassment, and (3) Result in unnecessary loss of property right. The classification decisions would be made on the basis of public policy, laws, legal interpretations, agency specifications and personal needs of individuals. However, the decisions would be expected to vary among record-keeping systems depending on the context in which the data are embedded, the amount of information and its intrinsic nature, the sophistication of the social values of the individuals involved, and the significance of personal attributes in the sub-culture involved. Among the examples of personal information that may be classified as "restricted," Comber cites political and religious preference, marital history, family attributes, ancestry and names of relatives.

A more detailed information classification system and sensitivity scales have been proposed by the British Computer Society:[4]

1. *Public*—Any personal information that is generally available in a listed form, such as various directories, biographic publications, etc.
2. *Published*—Information that is available but has not been collected, such as court records or hospital admission records. This category differs from "public" information in that if the individual involved does not draw attention to its existence, this information is not generally known.
3. *Confidential*—Information that is not generally available, although it is available and known to the individuals concerned.
4. *Secret*—Information that is not generally available, including the individuals concerned. Information in this category would be collected only under statutory authority or when authorized by the individual involved.

Within these four sensitivity categories there could be a more detailed sensitivity scale, such as illustrated in Table I. The authors of this classification system also point out that sensitivity of personal information varies with the circumstances and ideally each case should be determined according to precedents within the framework of legislation or professional codes of conduct, and that it appears not practicable to define degrees of sensitivity in any rigorous manner.

A very detailed catalog of classification of personal information items on the basis of sensitivity has been developed by Bing[5] from the point of view of Norwegian societal, legal and cultural concepts. The index contains some 400 data elements that are graded on the basis of three sensitivity levels:

1. *Normal aspects* (*GS1*)—General factual information about an individual's person, family, housing, property, employment, and other information in public record-keeping systems.
2. *Personal aspects* (*GS2*)—Intimate, detailed or specific information on an individual that could be used to make a social judgment about him as well as to obtain a detailed picture of his person, health, family, life style and views.

TABLE I—A Scale for Data Sensitivity (British)

Value Scale	Examples
0	Information collected and available, such as telephone books, professional listings
1	Selected general information (e.g., titles such as Miss, Mrs. or which indicate the marital status)
2	Public utilities account inquiry systems
3	Public information in schools
4	Vehicle licensing systems
5	Financial information (e.g., bank records; medical records)
6	More sensitive financial information (e.g., company finances)
7	Commercial secure information (e.g., trade secrets)
8	Confidential police records (e.g., records used by inquiry agents)
9	Police records relating to convictions
10	Secret information (diplomatic secrets; defense secrets)

3. *Disparaging and defamatory aspects* (*GS3*)—Information that could be used to form a moral or ethical picture of an individual, as well as information on especially sensitive health conditions or handicaps, ideological views and beliefs, law enforcement information, personal idiosyncracies and habits and evaluations of abilities.

The criteria for categorizing a data element depend, as in other classification systems that have been proposed, on considerations such as the individual and societal values, quantity of information involved, purpose of its collection and use, context and age of the information.

Sensitivity categories have also been proposed for various specialized record-keeping systems. For example, in the guidelines for record-keeping in public schools[6] the following sensitivity categories are proposed:

1. *Category A*—Official administrative records that constitute the minimum personal data on students necessary for the operation of the school (identification, attendance, academic work completed, level of achievement, emergency information).
2. *Category B*—Verified information of clear importance but not absolutely necessary (intelligence, aptitude and achievement test scores; health and family background; teacher and counselor ratings; verified reports of recurrent behavior patterns).
3. *Category C*—Potentially useful information, but not verified or clearly necessary beyond immediate use (legal or clinical findings, personality test results, unevaluated reports by teachers or counselors).

Specific administrative procedures are proposed for each category regarding access and dissemination, retention and use. For example, it is recommended that unless category C items are verified and, thus, moved into category B, they should not be retained longer than one year without discussing the reasons for this with the student's parents.

Finally, the following set of sensitivity categories has been proposed for criminal justice information systems:[7]

1. *Restricted*—Data that require minimum special security consistent with good security and privacy practices.
2. *Confidential*—Criminal justice information on individuals disseminated to criminal justice agencies, research reports derived from such information, and documentation of the information system itself.
3. *Highly sensitive*—Data that require maximum special security provisions and particularized privacy protection, such as criminal history information accessed by using other than personal identifying characteristics, arrest information without conviction, intelligence information, and computer programs and systems used for processing criminal justice information.

The lowest sensitivity category proposed in this system is "restricted" even though much of criminal justice information is public by statute. This is explained by pointing out that there is still a need to assure the integrity of such information.

A GENERALIZED CLASSIFICATION SYSTEM

The establishment of a standard classification system for controlling the use, dissemination and protection of personal information in record-keeping systems has the obvious benefit of clarifying for everyone concerned the level of privacy protection that can be expected and must be provided, and the consequences of not doing so. For each category would be specified the requirements for maintaining information quality, integrity and security; information handling and accountability procedures; personnel clearance criteria and procedures; information retention periods and classification criteria and procedures; and penalties for willful violations of these requirements. A framework for such a classification system is outlined below. It is discussed in more detail elsewhere.[8]

A very important consideration in setting up a classification system is the number of categories that are defined. Too many categories may make the use and implementation of the system too cumbersome and costly; too few categories may result in overclassification and excessive privacy protection requirements. Important considerations here are the number of sensitivity levels of personal information and the number of different dissemination restrictions.

Sensitivity levels

Personal information becomes sensitive when its uncontrolled dissemination may have adverse effects on the individual concerned and on his activities within his social group, or when it can reveal that the individual does not possess values expected by his family, acquaintances, those making determinations affecting him or the society. Two situations arise: the individual wants to limit circulation of the information, or the information is kept from the individual for "his own good" or the society's good. For example, the information may include an individual's past transgressions, views or associations or, in the second case, it may include results of medical or psychiatric examinations, or information on an ongoing criminal investigation of the individual.

The adverse effects of revealing personal information on an individual to others or, as the case may be, to himself, may range from a mild annoyance to physical harm or even loss of life. Between these extremes it is possible to define many other levels of adverse effects on the individual's physical and mental health and well-being, employment, family life, reputation, social life, and values. However, in order to keep the number of sensitivity levels small a scale of six categories is proposed in Table II. Shown are only the primary potential adverse effects of uncontrolled dissemination of information in each category; it is possible for adverse effects to escalate into the higher categories. For example, the release of information that results in a loss of self-respect may further lead to antisocial behavior, loss of employment and serious mental conditions.

Dissemination categories

Another consideration in setting up a classification system involves the restrictions that are placed on dissemination of the information by statutes such as the Privacy Act and the Freedom of Information Act, and by procedures adopted by the record-keeping organizations. The following possible recipients of information must be considered:

- The individual to whom the information pertains or those formally representing his interests (guardian, physician, lawyer, accountant). There are two aspects here—knowledge by the individual that a record of information is kept on him, and access to that information.
- Personnel of the record-keeping organization. There are two groups—those who have a specific need to use the information, and other personnel of the organization.
- Organizations with subpoena power, such as courts, grand juries, investigative committees at various levels of government.
- Any member of the general public who requests to see the information.

For each of the above, gaining access to the information is the principal consideration. However, for certain types of information the individual himself may need to be denied knowledge of the existence of a record on him, denied access to the content of the record, or both. For example, the Privacy Act of 1974

TABLE II—Sensitivity Scales for Personal Information

Category	Individual		Access granted to users		Subject to subpoena	Access to Public	Examples of information; information sensitivity levels
	Knows	Has access	Authorized	Others			
AS: Public (by statute)	Yes	Yes	Yes	Yes	Yes	Yes	Property tax rosters; Level 0 information
A: Public	Yes	Yes	Yes	Yes	Yes	Yes	Employee directory; Level 0 information
B: Limited, Official	Yes	Yes	Yes	Yes	Yes	No	Personnel records; Level 1 information
C: Restricted	Yes	Yes	Yes	No	Yes	No	Payroll records; Level 2 information
D: Confidential (by statute)	Yes	Yes	Yes	No	No	No	Social research data; Level 3 and 4 information
E: Sensitive (by statute	Yes	No	Yes	No	No	No	Psychiatric examination records; Level 4 and 5 information
F: Secret (by statute)	No	No	Yes	No	No	No	Organized crime investigation records; Level 5

exempts certain testing information from access by the individual, and the existence of certain criminal investigation records may be kept secret from the individual while the investigation is in progress.

Organizations with subpoena power can demand access to any record-keeping system which they consider important for their investigation and which is not provided a privileged status by law.[9] Information that is protected from subpoena includes the U.S. Census data and certain medical and psychiatric records. Other information granted statutory immunity from subpoenae in various states includes[10] drug abuse, alcoholism, and venereal disease records; information on victims of sex crimes, adoption proceedings, and illegitimacy records. However, personal information gathered for research purposes in social, behavioral and political sciences areas, and in education and psychology, is not provided with statutory protection against subpoenae and, as illustrated by recent cases,[11,12] the researchers' promises to keep the information confidential often have no substance. Hence, every classification category should also indicate whether or not protection against subpoena is provided.

Based on these considerations, dissemination control categories can range from "public" information which is accessible to anyone to "secret" information which is accessible only to authorized users of the record-keeping organization (the individual concerned neither has access to such information on him nor can he determine whether or not such a record on him exists). Table III depicts a classification system of six categories that should be suitable for personal information record-keeping systems both in government and in private sphere.

Clearly, the classification categories in Table III do not represent all possible combinations of access control restrictions. For example, it is conceivable that access to a particular type of information may be denied to the individual, but it may still be subject to subpoena power. In such cases the present classification system may have to be expanded.

INTEGRITY AND SECURITY PROVISIONS

Given a set of access control categories such as those defined in Table III, a set of requirements for assuring integrity and security of the information in each category can be specified. However, not every category may need a separate set of specifications. In Table III the categories AS and A are essentially the same, and so are the categories C and D (they differ only in whether or not information in these categories is subject to subpoena power). Thus, three or four levels of integrity and security provisions may be sufficient. In

TABLE III—Classification of Personal Information

Sensitivity level	Potential Adverse Effects on the Individual	Examples of Information Revealed
0	No appreciable adverse effects	Widely available, common information
1	Loss of respect in social sphere, loss of friends, loss of privacy and solitude	Remarks made in private; publicly available information not widely disseminated; information on views, preferences, leisure activities
2	Loss of reputation, recognition, social acceptance, self-respect, loyalty, competence	Information on political views, anti-social behavior, evaluative statements by the individual or others
3	Loss of economic security and opportunities, employment; disruption of family life	Information on medical and psychiatric treatments; sexual deviations; extra-marital affairs; evaluative statements by or about family members; criminal history
4	Loss of civil rights, imprisonment, serious effects on mental and physical health	Self-reported information on illegal or anti-social behavior; information on medical and mental condition; psychiatric evaluations
5	Loss of life or physical safety	Information that the individual is an under-cover agent for an investigative agency

any computer system there must be implemented a set of "basic" integrity, security and auditing procedures to prevent inadvertent interference of users with each other, accidental modification or destruction of information, and physical damage to the equipment.[8,13-16] Such a basic set of requirements should be sufficient for information in categories AS, A, and B (and to the sensitivity categories 0-2 in Table II).

A "medium" level of integrity, security and auditing is needed for information in categories C and D (and in sensitivity levels 3 and 4) since access to this information is limited to authorized users only and this information has a greater potential for adversely affecting the individual. The security and integrity provisions should include marking of the information items and records, and hard copy, either as "Restricted" or "Confidential"; establishing users' accountability for such information; implementing more sophisticated identification, authentication, and authorization procedures;[17] implementing audit logs; and strengthening the basic integrity assurance provisions.

A "high" level of integrity and security assurance would be provided to information in categories E and F (and for sensitivity levels 4 and 5). All files in the computer and all hard copy should be marked as "Sensitive" or "Secret" and stored securely; encryption techniques* should be used to protect information in these categories on removable storage media and in communication systems; sharing of the computer system with other computer activity should be limited, the system should be entirely dedicated to the record-keeping system in question, or (especially for sensitivity level 5) the information may have to be kept off-line or even in manual files; there should be full accountability of the users for handling information in these categories; sophisticated audit trails to trace file accesses to users and enhanced integrity control procedures such as change and error detecting codes should be implemented.

The above are only a set of general suggestions of what types of protection and integrity procedures might be used for the various information categories. In practice, the provisions adopted should reflect the specific circumstances of the record-keeping system and a thorough analysis of the security risk exposure of the information as well as a cost-benefit tradeoff. However, the methodology for risk assessment is still in the development phase at the National Bureau of Standards and elsewhere, and the cost of providing integrity and security is also known only in a very rough term.[20,21]

Classification policies and problems

Given a sensitivity scale and a corresponding classification system it is necessary to establish a set of standard criteria and a standard policy for classifying personal information items, records that contain several information items, and entire systems of records. Certain types of personal information used by the government can be classified directly on the basis of statutes that apply to the record-keeping system or to the information categories involved. For example, all personal information collected as part of the census are automatically "confidential," while property tax records are "public by statute" and psychiatric records are "confidential" or "sensitive." For other information not covered by statutes it may be necessary to first determine its sensitivity level and then to use this for making the classification decision.

One approach to standardize the assignment of sensitivity levels is to generate a handbook where, as suggested by Bing,[5] sensitivity levels are assigned to all personal information items that are known to occur in record-keeping systems (e.g., name, date of birth, amount of income, name of the employer, names of acquaintances, leisure time activities, etc.). A less detailed approach that is being considered for implementation of the Privacy Act of 1974 would assign sensitivity levels only to categories of information (e.g., identifiers, physical characteristics, employment history, evaluations, etc.) rather than individual information items and provide a list of these. In both cases there is the problem of deciding what sensitivity level should be assigned. Sensitivity is a highly subjective and context-dependent property of personal information—what one individual may consider very sensitive may be regarded with indifference by many others, and it is likely that there is a large range of sensitivity assessments for every information item. However, it would not be practical to assign the top sensitivity level of this range to each information item. Instead, a reasonable sensitivity level must be determined through the use of surveys and expert opinion.

A traditional approach to classifying a record that contains several information items that have already been classified, or for declassifying an entire record-keeping system, is to assign it to the highest classification found among its elements. Here, too complications may be found. For example, under some circumstances a collection of information items may be more sensitive than any one of the elements. In other record-keeping systems only very few information items may have a higher sensitivity level than the rest of the records and, thus, escalate the classification of the entire record-keeping system. The first case requires the development of an algorithm for increasing the sensitivity level of a record or record-keeping system as a function of the amount, type, sensitivity and uses of its elements. Research on this is yet to be done. The second case could be handled by using special security techniques (e.g., encryption) to reduce the sensitivity level of the information items in question, or by establishing a separate record system for these

* See papers on encryption in this Volume and References 17 and 19

information items. Other problems that must be tackled in assigning sensitivity levels to personal information and making classification decisions include automated classification of records and information derived from existing records, and downgrading of sensitivity levels and classifications as circumstances change.

Concluding remarks

Laws now in force require that privacy protection be provided to personal information in record-keeping systems maintained by the federal government, and by state and local governments in three states. Other states are expected to enact similar legislation, and pending in Congress is a bill, H.R. 1984, the Comprehensive Right of Privacy Act, which would extend privacy protection also to record-keeping systems in the private sphere.

It is necessary for effective implementation of these requirements to establish a standard sensitivity classification system for personal information such that for each classification level it is known what integrity and security assurance must be provided, what information handling practices must be followed, and what penalties apply for non-compliance. Information items can then be assigned into appropriate categories on the basis of their potential to adversely affect individual data subjects, and on the basis of access and dissemination limitations that may be required by law. One such classification system and a sensitivity scale is proposed in this paper. Important questions still being researched deal with criteria and policies for determining the sensitivity and classification levels of information items, records and record-keeping systems.

REFERENCES

1. *Privacy Act of 1974*, Title 5, United States Code, Section 552a (Public Law 93-579), December 31, 1974.
2. *Freedom of Information Act*, Title 5, United States Code, Section 552, 1967.
3. Comber, E. V., "Management of Confidential Information," *AFIPS Conference Proceedings*, Vol. 35, 1968 FJCC, pp. 135-143.
4. Ellis, L. (Ed.), *Privacy and the Computer—Steps to Practicality*, British Computer Society, London, July 1972.
5. Bing, J., "Classification of Personal Information With Respect to the Sensitivity Aspect," *Databanks and Society*, Universitetforlaget, Oslo, 1972, pp. 98-150.
6. *Guidelines for the Collection, Maintenance and Dissemination of Pupil Records*, Russell Sage Foundation, New York, 1969.
7. "Data Sensitivity Classification," *Criminal Justice System*, National Advisory Commission on Criminal Justice Standards and Goals, Washington, D.C., January 1973, pp. 128-130.
8. Turn, R., *Privacy and Security in Personal Information Databank Systems*, R-1044-NSF, The Rand Corporation, Santa Monica, March 1974.
9. Nejelski, P. and L. M. Lerman, "A Researcher-Subject Testimonial Privilege: What to Do Before the Subpoena Arrives," *Wisconsin Law Review*, Fall 1971, pp. 1085-1148.
10. "The Computerization of Government Files: What Impact on the Individual?", *UCLA Law Review*, September 1968, pp. 1371-1498.
11. Kershaw, D. N. and J. C. Snell, "Data Confidentiality and Privacy: Lessons from the New Jersey Negative Income Tax Experiment," *Public Policy*, Spring 1972, pp. 261-269.
12. Walsh, J., "Antipoverty R&D: Chicago Debacle Suggests Pitfalls Facing OEO," *Science*, September 19, 1969, pp. 1243-1245.
13. *Guidelines for Automatic Data Processing: Physical Security and Risk Management*, FIPS Pub. 31, National Bureau of Standards, Washington, D.C., June 1974.
14. *Computer Security Guidelines for Implementing the Privacy Act of 1974*, FIPS Pub. 41, National Bureau of Standards, Washington, D.C., 1975.
15. *AFIPS System Review Manual on Security*, AFIPS Press, Montvale, N.J., 1974.
16. Turn, R. and W. H. Ware, "Privacy and Security in Computer Systems," *American Scientist*, March-April 1975, pp. 196-203.
17. Saltzer, J. H. and M. D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, September 1975, pp. 1278-1308.
18. Turn, R., "Privacy Transformations for Databank Systems," *AFIPS Conference Proceedings*, Vol. 42, 1973 NCC, pp. 589-601.
19. Feistel, H., W. A. Notz and J. L. Smith, "Some Cryptographic Techniques for Machine-to-Machine Data Communications," *Proceedings of the IEEE*, November 1975, pp. 1545-1554.
20. Goldstein, R. C., *The Cost of Privacy*, Honeywell Information Systems, Brighton, Mass., 1975.
21. Turn, R., "Cost Implications of Privacy Protection in Databank Systems," *Data Base*, Spring 1975, pp. 3-9.

Philadelphia justice information system

by IRVING J. CHASEN
PJIS Project
Philadelphia, Pa.

ABSTRACT

The Philadelphia Justice Information System (PJIS) concept was designed via a joint study with a major computer manufacturer. Functions—generally threefold: (1) Monitor and Control—to act in an active and directive nature to insure completion of all tasks, to automatically issue reminders for tasks not completed on schedule, and to make administrative contacts for corrective action until all tasks are completed, (2) Data Collection—merger of and improvement upon existing data processing systems and (3) Communication—to make information available upon request to all cooperating agencies. All agencies related to the criminal court system (e.g., Public Defender, Clerk of Court, District Attorney, Probation Department, Prison) will share PJIS Data Base with Courts and Police. PJIS Data Base will be result of merger of two extensive existing Criminal Justice Data Processing Systems—Police and Courts. Goals of single system—(1) eliminate redundancy, (2) reduce overall costs, (3) attempt to accomplish conflict-free scheduling, (4) assure back-up capability, and (5) more effective coordination of man and machine resources within the courts and agencies of the Philadelphia Justice Information System to increase fair and speedy case disposition. Guidelines are being drawn to build in protection of the individual's right to privacy.

The Philadelphia Justice Information System (PJIS) is a computer-based system designed to serve the information needs of all the agencies of the Justice System. These agencies include the Police, Courts, District Attorney, Public Defender, Prisons, and Probation Department. The PJIS concept was designed via a joint study conducted with a major computer manufacturer between 1971-1974. The system is being implemented by the PJIS Project which is currently funded through a LEAA Discretionary Grant. The focal point of this system will be a central computer with associated programs, centralized data base and communications network.

Currently, most of the planned functions of the system are operational. There are automated systems now serving the Courts, the Prisons (PRINS), the Police (On-Line Booking) and the District Attorney's Management Information System (DAMIS). The next step will be the incorporation of the existing applications into one single data base serving the various agencies with a minimum of data redundancy.

PJIS is oriented around a single Data Base, a central repository of Justice System information. The justice agencies share this common fund of information in performing their separate but related tasks. Each agency has responsibility for entering information into the Data Base, and personnel in each agency are permitted access to information for which they have a legitimate need. Data Base information is organized so that each user can refer to it and analyze it according to his own needs. Thus, each agency may—quite realistically—view the computer as being its own. The single Data Base allows ready access to accurate, current, comprehensive information. Persons in each functional area can feel confident that the information they are using is consistent with that being used in other functional areas.

Beside the specific information services it provides, the Data Base also has a more subtle, though important, value. The mere existence of this shared resource should have an integrative impact on the justice community, since it enhances the single-process idea and helps to create an atmosphere in which agency interdependencies become more apparent and increased cooperation is seen to be both possible and mutually profitable.

Each of the Justice agencies is directly linked to the shared Data Base by a network of computer terminals. These devices, equipped with a video display, a keyboard, and a printer are dispersed throughout each agency's facilities. Each terminal is a window to the Data Base: each makes the Data Base continuously present to authorized persons.

It is through these terminals that each agency enters its information into the Data Base. It is also through these terminals that information is retrieved, in one of several ways:

(a) INQUIRIES: The computer system will provide immediate response to requests for information; responses are in the form of printed copy or video display or both.

(b) NOTIFICATIONS: The computer system will issue unsolicited printed messages whenever it needs to inform someone of a new responsibility (Task) assigned to them or when it is necessary to quickly distribute newly received information about a particular incident, person, or case.

(c) DOCUMENTS: Whenever practicable, the computer system *will* create file folder documents on printers located within each agency. This method of document distribution, when used, replaces the exchanging of carbon copies.

(d) REPORTS: Statistical analyses, trial listings, and other reports of that type *will* be created on printers located within each agency rather than being printed at a central location and distributed manually.

The system provides a common statistical base which is used to prepare analyses for management use. While individuals with different viewpoints may question the significance of certain statistics or may interpret them differently, everyone in the justice community will be using figures derived from a single source—the PJIS Data Base.

Because each agency is an active participant in PJIS, the computer system has the necessary information to assist in scheduling judicial proceedings. It knows scheduling rules and case data, and it maintains availability schedules of courtrooms and of Justice System professionals. The system uses this information to recommend conflict-free dates and locations each time someone indicates a need to schedule a courtroom event.

Thus, the system "closes the loop" in its ability to assist in controlling Justice System activities. It actively participates in scheduling the majority of events, and then, as outlined earlier, the system actively monitors the accomplishment of tasks necessary to make those events occur as scheduled.

File folders will continue to be used in PJIS, but the method of creating and distributing many of the documents for those folders will change. For example, a single document showing the known facts about each new case *will* be printed onto a single document at the time a case enters the Justice System. In addition, a follow-up document, showing data accumulated since the case entered the system, will be printed each time a case is due for a court hearing. Interested agencies receive these documents on printers located on their own premises. These consolidated information sheets serve a dual purpose: they keep case folder information current while using a minimum number of pieces of paper.

In summary, PJIS interacts with operational and administrative personnel in the performance of their daily activities. Through terminals, it prompts them, and responds to their requests for immediate information. As a result of the system's active participation in events as they occur, the Data Base always reflects a current and comprehensive picture of the Justice System and the status of every incident, person and case. Thus, the system is able to respond to requests from management and professional staff for information on demand.

The Data Base integrates Justice System information into a cohesive collection of related data. It is the central repository for two general categories of information:

> Operational Data—supplied by and used by Justice System personnel directly in the course of day-today operations. This "live" information represents the facts of the Justice System, such as statistics, historical records, and details concerning current cases, incidents, persons, and scheduled hearings.
>
> Control Data—supplied by Justice System management and used by the computer in its decision-making processes. This relatively static (but easily changeable) information specifies available resources, system policy and operating guidelines. It includes such things as a description of courtroom facilities, availability schedules for police, judges, attorneys (for use in scheduling); rules for granting continuances; allowable intervals between successive stages of defendant processing; and who is permitted access to what data. This kind of information will be stored in Data Base tables rather than being embedded within numerous application programs, a scheme which permits easy modification by authorized persons and enables management to readily exercise control over system operation.

The communication network links remote locations with the computer and the data base. Online terminals equipped with video display and printing capabilities will be placed at strategic locations to allow information retrieval on demand, and in some cases, to also permit keyboard entry of data into the data base. In addition to the local communications network, the system also includes provisions for exchanging information directly with State and National Law Enforcement data bases (the CLEAN and NCIC Systems).

The independent justice agencies jointly use the system facilities. These users effectively share a common fund of information, the PJIS Data Base, in performing their separate but related tasks involved in processing defendants through the Justice System. In this cooperative effort, each has responsibility for entering data according to fixed rules, and personnel in each agency are permitted access to data on the basis of their need to know.

Data Base information is organized so that all users can refer to it and analyze it according to their own needs. The Data Base allows maximum accessibility to accurate and current information by authorized Justice System personnel, so that each functional area can perform its tasks with the knowledge that the information being used is consistent with that used by other functional areas. While PJIS is intended to aid operating personnel, it also is the means by which management personnel can maintain a comprehensive awareness of overall system status—who is presently responsible for doing what, when, and where; what are the sources of recurring problems which disrupt the system; and in what departments information-handling operations need revision.

The ultimate goal of any information system is to make reliable data readily available when, where, and in the form it is needed. Accordingly, the design of modern automated systems, including PJIS, is directed away from paper movement and toward the use of data bases and online terminals for recording, retrieving, and exchanging information between system users.

PJIS collects, stores, and distributes data concerning three type of entities:

(a) Incidents: The matter for which a subject (person) has entered the Justice System. In most cases, it is a police incident, but incidents may also be originated by other means, principally by the District Attorney's Office which handles private criminal complaints and investigative grand juries.

(b) Persons: defendants processed through the Justice System. At various points in the processing cycle, an individual may be referred to as the subject, suspect, arrestee, defendant, offender, prisoner, parolee, juvenile, detainee, client.

(c) Cases: court units of scheduling.

PJIS could be referred to as an incident-tracking system, or a defendant-tracking system, or a case-tracking system—and all such descriptions would be partly correct, because the system does track all of these entity types rather comprehensively.

PJIS plays an active role in the justice process; it is far more than a passive record-keeping and reporting system.

The control capabilities incorporated in the computer system design give it the ability to:

Assist in scheduling judicial proceedings
Coordinate tasks and events
Monitor the accomplishment of tasks
Manage the distribution of information
Assist data entry operations
Provide for data security
Assist with enforcement of Justice System rules

Each of these capabilities operates together in a systematic manner. Numbered steps in the following narrative refer to correspondingly numbered items in Figure 1.

(1) Major events are scheduled by persons, not by the computer system. However, where the event is a judicial proceeding, the computer system assists in establishing a conflict-free date and location for the hearing. It is able to assist in this way since it maintains availability schedules for courtrooms and for key participants in the system (Judges, Attorneys, Police witnesses).

(2) Each time a major event is scheduled, the Coordinate function is informed. The Coordinate function knows what standard tasks must be performed before, during, and immediately after each type of major event. Since it has been told when a particular event is to occur, and since it also knows how many days before or after an event that each task must be performed, the Coordinate function can easily calculate the date by which each routinely-performed task must be completed.

(3) The Coordinate function passes to the Monitor function information about each task which must be performed. This information specifies (a) the identification of the entity (incident, person, case) for which the task is to be performed; (b) a description of the task; (c) who is responsible for performing it; (d) the time by which it must be accomplished; (e) the time when the Monitor function should first check to see whether the task has been completed. The task is placed into a Pending Tasks List which is continuously reviewed by the Monitor function.

(4) Persons may inform the system at any time that special tasks, not routinely performed for every entity, must be performed for a particular one.

(5) Each time someone informs the system that a special task is needed, the Coordinate function handles it essentially as if it were a normally required task and places it into the Pending Tasks List. In addition, the Coordinate function usually issues an immediate notification to inform the responsible Justice Unit of the non-routine task to be performed.

(6) Persons spread throughout the Justice System are continually entering information into the Data Base through remote terminals equipped with keyboards and video displays. In doing so, they operate in a conversational mode with the computer system which: (a) performs a security check to uniquely identify each operator and to ensure that persons entering particular kinds of data are authorized to do so; and (b) guides the operator through successive steps of data entry to ensure, to the maximum extent possible, that information being entered is both complete and logically con-

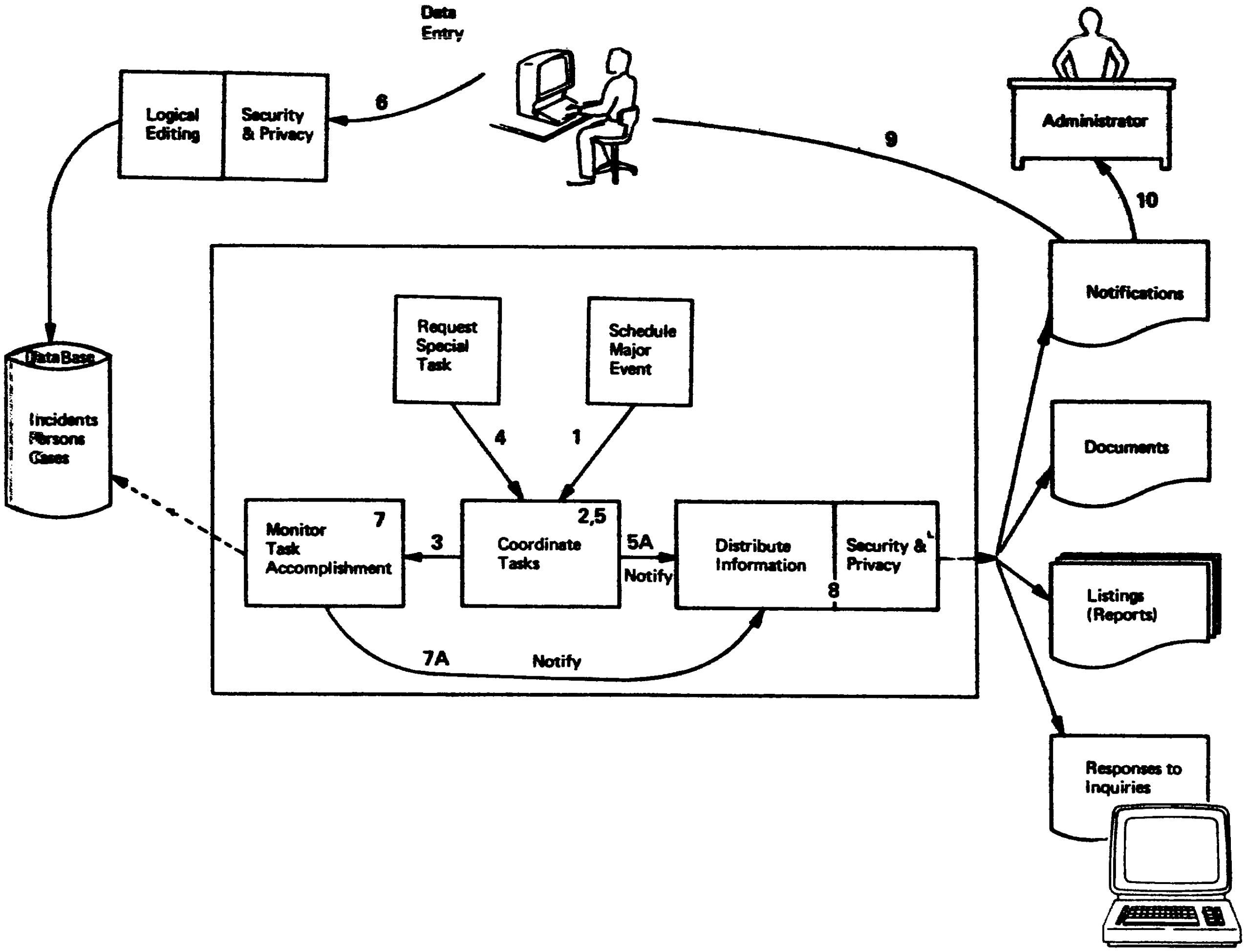


Figure 1—Overview of "principal capabilities"

sistent. Most input transactions fulfill a requirement specified in the Pending Tasks List. That is so because, ideally, the system (via the Pending Tasks List) anticipates all input transactions so that it may remind responsible persons when input has not been received when due. It should be noted all input transactions are automatically logged and time-stamped within the computer system. This information is a useful aid in constructing audit trails.

(7) The Monitor function continuously selects items from the Pending Tasks List which have become due (each item carries with it the date and time at which it should be reviewed by the Monitor function). The Monitor function then looks at the Data Base to see whether a task due for completion actually has been accomplished. Where it has been, no further action is required. But where it has not, the Monitor function issues a notification to the appropriate Justice Unit reminding them that a task for which they are responsible has become due. The item is placed back into the Pending Tasks List for another review a short time later. If the task is not performed after some period of time, a notification will be issued to a responsible administrator to inform him of the exceptional situation.

Of course, the Monitor function also keeps statistics on the types of tasks it is monitoring and the number of times those tasks are or are not completed satisfactorily. These statistics will be used by administrative personnel to evaluate procedures and to determine the root causes of Justice System problems.

(8) The Distribute function manages the distribution of all information produced by the com-

puter system. This material falls into four general categories:

(a) Notifications: messages about a single entity. They are always created by the computer system (not by persons outside) either to request that someone perform some specific task or simply to provide new information as soon as it becomes available. Whenever possible, notifications are transmitted from the computer directly to the Justice Unit for which the information is intended.

(b) Documents: computer-printed forms containing facts about a single entity. They are generally intended for insertion into file folders for reference purpose.

(c) Listings (reports): computer-printed forms containing information about groups of entities. They include such things as trial listings and statistical analyses.

(d) Responses to Inquiries: the computer system responds immediately to requests for information on demand (subject to privacy restrictions). These requests are made through terminal keyboards at locations remote from the computer site. Computer responses are in the form of video displays or printed material or both. PJIS is heavily oriented toward distributing specific information in this way rather than through the regular production of voluminous printed reports. Most information that is regularly published in printed form is also available through terminal inquiry as well. It should be noted that all outputs are logged and time-stamped within the computer system.

(9) For the present discussion, attention is focused on notifications, specifically those created by the Coordinate and Monitor functions. The Coordinate function creates notifications to inform Justice Units that special tasks, which are not routinely performed for every entity, must be performed for a particular one. Every task requires entry of information into the Data Base, even if it is only an affirmative report that some task, known to the computer system, has been performed.

The Monitor function creates notifications to remind Justice Units that tasks for which they are responsible are due but have not yet been reported as accomplished. Before requesting corrective action by administrative personnel, the Monitor function usually checks again a short while later to see whether, as a result of the prompting, overdue tasks have been accomplished.

(10) The Monitor function also creates notifications to inform administrative personnel that the deadline for accomplishing a certain task has passed, and some action on their part is required. For example, if a psychiatric report will not be available as ordered, a request for continuance of a sentencing hearing may be necessary; or, if no outcome has been reported for a particular hearing, even after the responsible unit has been reminded of its absence, an administrative manager is informed so that he may research the problem.

An illustration of the operational services is provided in Figure 2. This example illustrates the flow of information that is contained in Agency file folders and that is also entered into or retrieved from the computer system data base. The operations shown occur in stages or over many stages but are consolidated for the purposes of this example.

The first entry into the system occurs at the Justice System Entry stage with a good deal of the information recorded at booking. The Static Information Sheets are printed on the agencies' administrative terminals from which they create their file folders. The agencies' File Folder Control Unit records the location and party responsible for file folders checked in or out.

Updates to the file folders are accomplished with the Volatile Information Sheets which are produced some time prior to each hearing. The information that changes or adds to the Volatile Information Sheets is stored on the Data Base. The Data Base is constantly being updated by all agencies over remote terminals which are also used to retrieve required system information.

The Data Base is updated at the hearings by the Court Clerk with case proceedings such as judicial orders, case disposition, next hearing schedule, and attorney appearances. Each agency's audit unit then compares the system information recorded in the file folder at the hearings.

The flow is then repeated for each additional hearing until the case is completely disposed. The data base information and the file folder for that case are then purged from active storage but retained in high density storage for historical purposes.

The above description applies to each agency concerned with file folders in the process of the hearing stages. Although only one agency illustrates the complete flow, it actually occurs in parallel with multiple agencies.

OVERALL SYSTEM BENEFITS

Effective data recording

By requiring information to be recorded only once and at its source, the system provides the following benefits:

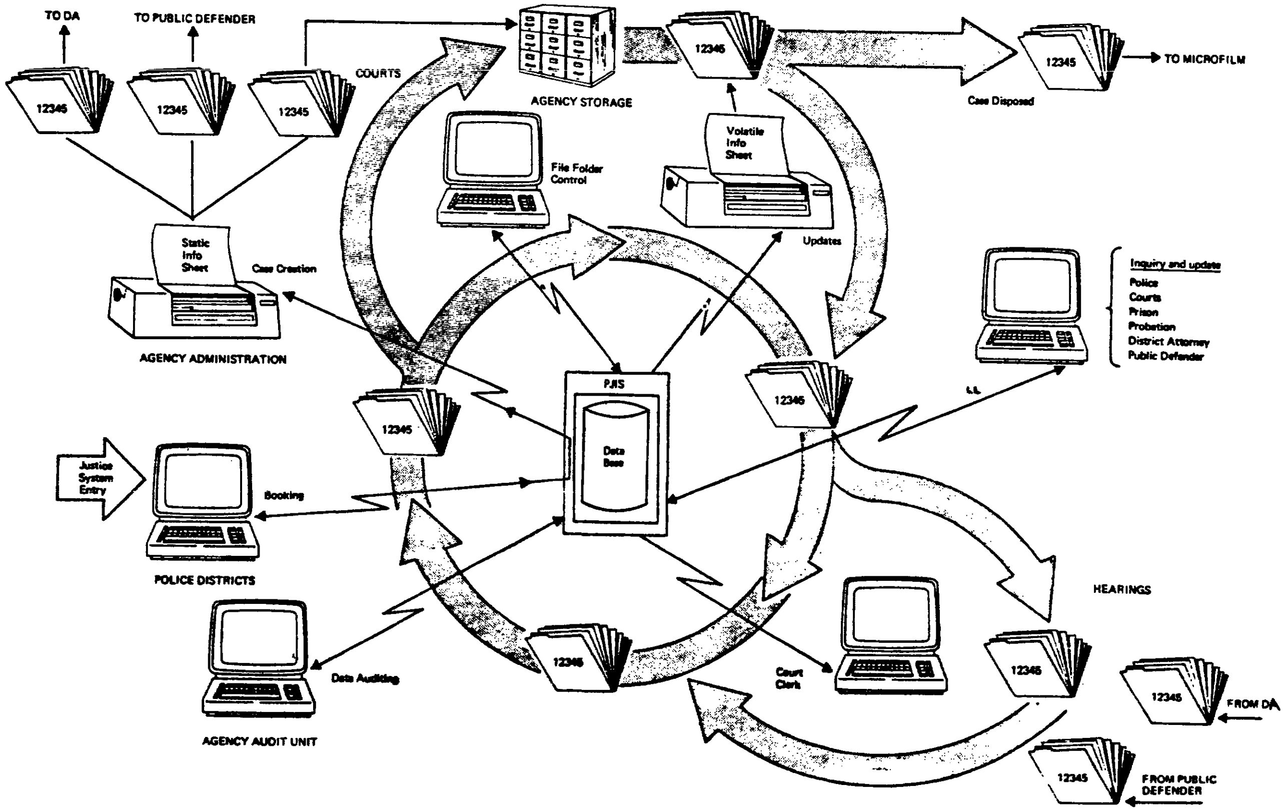


Figure 2—Conceptual Information Flow

1. Complete entry of all data related to each transaction
2. Accuracy of data entered
3. Reduction of redundant effort
4. Improved auditing and accountability
5. Reduced paper volume
6. Improved consistency of data

Improved security

PJIS exercises positive control over attempts to access information thus providing greater security than is possible with the existing information systems in Philadelphia.

Improved resource management

1. Flexible resource allocation
2. Increased personnel productivity
3. Better performance evaluation
4. Improved program planning and control
5. Management by exception
6. Concise management reports

Improved communications

1. Intra-agency communications
2. Inter-agency communications
3. Distribution of pertinent notices

REDUCED FILE MAINTENANCE

Reduction of continuances

1. Assistance with conflict-free scheduling
2. Optimum courtroom loading
3. Monitoring preparatory tasks

IMPROVED FEEDBACK AND FOLLOW-THROUGH ON ALL ACTIONS

Improved public image

Responsiveness to change

1. New applications
2. Changes in law
3. Policy and procedure revisions
4. Data Base content changes

Computers in architecture

by GENEVIEVE GREENWALD-KATZ
Max O. Urbahn Associates
New York, New York

ABSTRACT

The union of computers and architecture has not come about in spite of glowing predictions over the last ten years. This paper describes several of the reasons why this may be so: the complexity of the profession, the inattention as to how the architects actually design and the lack of top management dedicaton. The current state of the architectural profession is explored and why the architect's entry into new areas should spark a demand for computer technology is discussed.

INTRODUCTION

Over ten years ago, articles extolling the wonders of computers and predicting, in glowing terms, their impact on architecture began to appear. What has happened to this prediction—why has it failed? To understand this turn of events, we must look both at computing, and at architecture itself. Perhaps there, we might find clues regarding the introduction of new techniques into old professions that might even have more universal implications. What is the state of architecture today and of computer utilization in architectural practice? The architectural profession, as we know it, is going through a major upheaval. The recent deterioration in the economy has had a strong negative impact. Statistics from the New York Chapter of the American Institute of Architects state that architectural commissions have declined 55 percent from the year 1972 to 1973 and dropped an additional 50 percent in 1974. Reflecting this, the employment level shows a drop of 75 percent from 1969.

Some architects are now beginning to ask the question . . . how did we get to be so non-essential? In a time of rapid technological change and increasing costs, long established practices must be reevaluated in order for the profession to continue its existence . Architects are still designing buildings by hand—row upon row of people bending over drafting tables draw the details of each building.

In situations where resources are limited, the architect must be able to explore alternatives to building, to financing and to forms of energy utilization. With costs escalating, he must solve problems that are more complex and solve them faster than he ever has before. The architect must recognize these new needs and adapt his skills to solving these new problems.

Shrinking staffs and lack of work, however, is not being experienced by all firms. There are some firms that are expanding their offices. Some who are expanding currently are those who are using computers. It is not the "magic" of the machine that is doing this, rather, the computer can be seen as being symbolic of the architect's ability to learn new skills, handle large amounts of data and apply his expertise to the broad areas surrounding that of building. Let us look more closely at this phenomenon.

HISTORY

The coupling of computers with architecture began about ten years ago. While computers had been in use long before that, the period between 1963-1966 was marked by a significant development in computer graphics.

In 1963, direct-view bistable storage tubes were first used in terminals. That year, at the Spring Joint Computer Conference, Timothy Johnson and Ivan Sutherland announced their development of Sketchpad. That program made it possible for the architectural designer to input pictorial data into a computer by drawing on an electronic tablet. The drawing was displayed on a C.R.T. and could be modified with a light pen. The Rand tablet (the device used by Sketchpad) made its appearance at the Fall Joint Computer Conference in 1964.

Responding to the new advances being made, the First Boston Architectural Center Conference, held in December of 1964, had as its theme, "Architecture and the Computer." It was well attended and included many of the persons who were to make advances in computers for architects. Articles linking computers with architecture began appearing in architectural journals and trade magazines. Predictions ranged from computer takeover and subsequent dehumanization of architecture to an office where the computer would be the powerful servant of the architect, relieving him of all tedious tasks and leaving him free to spend all his time designing and dealing with high level abstractions. The dire predictions of computer takeover have not occurred and fears along that line have been quieted. Articles anticipating the widespread use of computers by the architectural profession however, continue unabated.

The time frame for this transformation was always within the next ten-year range. The ten years are now up. The tone of the articles has changed from enthusiasm about the use of the computers to bewilderment as to why the architectural profession has snubbed the computer, and finally to philosophizing that perhaps architecture and computers are totally incompatible. But what actually is happening?

I do not believe that architecture and computers are incompatible. However, I do feel that many of the people who would like to see computers being used more by architects:

(a) Do not understand what motivates the architect.
(b) Do not have a grasp of the complexity and variety of disciplines which the architectural profession encompasses.
(c) Have no idea as to how architectural offices function and in turn, how they effect how the architect works.

Furthermore, advancement in computer technology takes place primarily in the university, and many of the students are only remotely aware of how architecture offices work. Many of the programs developed in the academic environment are directed more towards the purpose of understanding the discipline of architecture than towards advancing the practice of architecture. I am not here questioning if it could or should be otherwise, but trying to explain why some of the "advances" do not make it to the office.

HOW WIDESPREAD IS THE USE OF COMPUTERS

First of all, the architecture profession is a small one. Figures from the 1970 U.S. Census show the comparative size of various professions:

Architects	57,081
Engineers	653,925
Lawyers	264,752
Physicians	280,557

A tally done by the American Institute of Architects from the state rosters of registered architects lowers the figure to 45,000. (The difference between the two figures can be explained in that "registered" means holding a license to practice, while in the U.S. Census "architect" referred to those people who stated that as their professional status).

While it has been the larger architectural firms that have gone into computers, the profession is characterized by an organization into small offices. Approximately 40 percent of the architectural firms are sole proprietorships. It is estimated that 90 percent of the firms employ six people or less. So the small number of actual users is explained in part by the logistics of the profession.

It is difficult to estimate the actual number of architectural firms using computers. The New York Times (August 29, 1971) estimates that of the 15,000 design firms in this country only 200 use computers. The 15,000 probably represent purely architectural offices and most likely includes the sole proprietorship type of office. To get some idea of the number of large firms, we can refer to The Engineering News Record which publishes a list each year of the top 500 design firms. "Top" refers to dollar billings and the 'types' of firm include architect-engineers, engineers-architects, consulting engineers, architects and design-contractors. Of the total number of firms listed about half are involved with architecture.

The issue for May 21, 1970 also included information as to the use of computers by these firms. Excluding design-contractors, "357 of the 460 top design firms reported the use of computers in their operations. Of the 357 firms, 32 percent rent their equipment, 29 percent use a time-sharing system and 13 percent own their own hardware. The balance used varieties of rental and time-share plans."

While the established pattern has been for the larger architecture firms to be exploring the use of the computer, there is growing evidence that smaller firms are also beginning to utilize computers. One of the reasons for this transition could be the lower cost of hardware. It is now possible to buy a small, but essentially complete computer with one high level language (Basic) for under $10,000. Ten years ago comparable equipment would cost between $30-50,000. Leasing a terminal in 1968 cost $100 per month while now terminals can be had for $50. Time sharing costs ($10-15 per hour) are essentially the same as they were in 1968.

Smaller firms have an easier time than larger ones in adapting to the changes brought about by the introduction of computers. Organizational changes and even new directions are not only simpler to effect, but often the younger members of the firms have less invested in the established way of doing things. There is also evidence that the use of computers by small firms differ from that of large firms. With large firms, computer usage is more likely to be only a small part of the office routine and only used by a small fraction of the staff. When small firms use the computer, the whole office may well be organized around its use. The entire staff is encouraged to put "hands on" the terminal. It is this "hands on" approach that allows an office to explore the tool. This exploration can lead to the application of computer technology to various areas of architecture. The important thing here is that the people working in the field are the ones who should begin to develop the programs.

In order to better understand how firms make use of computers, the New York Chapter of the American Institute of Architects has recently become interested in what motivates a firm to go into the use of computers. Some of the reasons given were:

(1) The approval of the systems approach and the methodology behind computer use.
(2) The need to process vast amounts of data in short time.
(3) The entry into a new discipline which has not yet developed a methodology.
(4) The fascination with the process and a feeling that this must be the future direction for architecture.

This last reason is essentially a commitment to architectural research.

Some of the reasons are frankly window dressing—ways of impressing a client with the firm's grasp of the new technology. But one thing they all have in common is that using computers involves a top management decision. It is the eye of the vice president that this tool catches. It involves top management decisions as to new directions for the firm. Most certainly it will involve a considerable investment and (for the success of the venture) it must have management's continuing active interest, often in the form of a single enthusiastic partner. This last factor explains, in part, why the various branch offices of firms differ widely in computer usage. In one firm, one office uses sophisticated graphics and computer-aided design programs, while in another branch, changes to specifications are done by the architect dictating to a secretary who is seated at a terminal. At one time it was thought that computers would make inroads into the architectural offices by way of the accounting departments. Unfortunately, once into the accounting office, it rarely gets out into the design section.

HOW MIGHT NEW AREAS OF INTEREST INCREASE COMPUTER USAGE?

A new factor which may cause an increase in the use of computers by architects is the entry by the architect into fields which are very compatible with computer usage: information manipulation, numerical calculations, linear programming and graph theory. Because of the decline of building, architects are exploring some of the other areas which are related to the building process. The architect is now looking into the processes which are active before the building is designed, such as economic analysis, building programming, establishment of standards; processes active after a building is built, such as evaluation; and processes involving not-building, such as renovation. "Programming" for an architect is the assembly of the detailed requirements for a project into a coherent form, not the writing of computer procedures. Some of the more specific areas where computers are beginning to be of assistance are the following. It will be interesting to see if in moving into new areas, the architect accepts the new technology.

LIMITED RESOURCES PLANNING

A decade ago 'limited resources' was a concept that architects did not have to concern themselves with. The recognition that earth's resources are finite demands that the profession deal responsibly with energy, time, money, land and space. Today it is irresponsible for an architect to plan hospitals without taking into consideration the logistics of health care delivery, population trends and availability of technical staffing. There are programs utilizing the techniques of linear programming which are useful in this area. They allow the architect to consider the effects of a number of variables operating at the same time to arrive at an optimum solution for that situation. Graphic output aids in understanding how the various factors operate.

ECONOMETRIC MODELS

Just as architects use physical models to explore design solutions, they will also use econometric models to help them with the many other factors influencing design decisions. Life-cycle costing, value analysis and cost benefit analysis are three current methods in favor with both the architect and his more sophisticated client. Computer techniques are extremely useful here because of the iterative process of the analysis. The analysis may cover a wide range of items, such as building components, systems, services and social benefits. Solutions are arrived at by combinations and recombinations of the variables. If many items are to be considered, the analysis by hand is extremely tedious and slow.

THEORY OF DESIGN

Not a new area but clearly one of the more favored. The push to demystify the design process and to understand how the architect evolves his solution is more in the purview of the architect-computer-researcher than of the architect-teacher. In order for computer programs to be able to assist the architect they must be designed with an understanding of the process which they are assisting. A comprehensive study of the design activity in architecture has been carried out in the Department of Design Research at the Royal College of Art in London by a team of researchers headed by Patrick A. Purcell. The study, "Analysis of Architectural Design Activity in the Working Environment" attempts both to develop a model of design activity and to evaluate computer-aided design techniques. The study begins to define clearly the points in the design process when the architect needs information and how he incorporates various types of information into his "internal representation." Insights derived from work done on design corroborate the architect's feeling as to the importance of this area. Design and planning may comprise about one half of the total time of the entire job. In complex building types, such as hospitals, economies in design and planning time can save a building from being obsolete upon completion.

Attempts to deal more objectively with the design process are being made by design offices. Governmental agencies and corporate clients, with increasing frequency, are asking for an "objective" analysis of the design. Cluster analysis and hierarchical composition, and resolution are some of the techniques being used to arrive at decisions. However, sometimes what is passed off as an objective scientific analysis of the problem is simply putting numbers on subjective decisions.

BUILDING PROGRAMMING

Architectural firms which have experience in a particular building type are eminently qualified to write programs for that type of building. Most firms have not developed techniques for saving and reusing their data and so, much of their valuable experience is only partly carried over to the next job of type.

Non-architectural firms which have become involved with preparing "Building Programs" have developed sophisticated data gathering and data processing techniques. Architectural firms, in general, have been slow in utilizing these techniques. And yet, the application of these techniques could amplify the architect's experience into a powerful programming tool. There is also an increasing demand for pre-programming analysis of data relevant to the proposed project and this analysis is becoming so complex that perhaps computerization is the only rational way to do it. Feasibility studies are also used to make some assessment before proceeding to the long and expensive process of writing a "building program."

FACILITIES MANAGEMENT

Different from construction management and more suited to the architectural firm with some expertise in general management, this area covers the comprehensive management of the existing and future building activity of a large facility. Real estate, financial and building cost data are manipulated in various ways in order to arrive at a most beneficial mix for the client. Again, it is necessary to explore a large range of alternatives.

Architecture contains a vast assortment of skills and disciplines which are necessary to produce a building. As the profession adapts to new needs, new skills will have to be acquired. The pattern of growth is outward, in the direction of higher order generalities such as the understanding of the socio-economic processes which affect the building, rather than towards areas which are more specific and circumscribed. Perhaps this fundamental outlook underlies the fact that there has been no great movement by architects, even in a time of unemployment, to go into the field of engineering.

ARCHITECT-COMPUTER INTERFACE

In designing an architect-computer interface one should be aware that the architect feels differently about different areas.

(1) There are things an architect cannot do or cannot do well, but which he is interested in doing.
(2) There are things which he doesn't want to do.
(3) There are things which he doesn't want to do—but he has to.
(4) There are things he does well in which he feels he doesn't need any help.

1. Things an architect cannot do or cannot do well, but which he is interested in doing. Computer assistance in this area is welcomed by the architect because with it he can attempt areas previously inaccessible. An example of this type is Christos Tountas' 3-dimensional tent construction program. In it the designer specifies the anchor points in plan and elevation and the computer generates the shape of the tent in which the stresses will be equally developed. The program will then provide load analysis and coordinates of all the nodes. The designer can, in addition, specify various elastic properties to the edge cables and because the system is interactive, the computer will redraw the resultant shape of the tent. This technique for designing tents far surpasses the method presently used—the construction of actual models.

Any program which allows an architect to design curved forms as easily as he does in straight lines would free him from the constraints of his straight line geometry. It would assist him in conceptualizing shapes and spaces which are currently too difficult to draw.

Simulation programs showing movement of material and people through a building would also be useful. The architect, while he can visualize space easily, finds it difficult to visualize continuing processes through the space. Since he must make design allowances for them, visual representation can be very helpful in understanding the problem.

2. Things which he doesn't want to do. If a program is provided that would simplify these tasks he probably would still rather someone else did them. An example of this are programs for structural analysis. Regardless of how easy it becomes, on any large project the architect will still want the engineer to do it.

3. Things which he doesn't want to do—but has to. These are areas of his responsibility like code compliance and specification writing. The simpler the process, the better. Moreover, it is not what the masterbuilder will get involved with as much as the guy in the back room. Furthermore, for a firm to buy a computer program, it will have to be cost competitive with their consultant.

4. Things which he does well, such as design, and feels he doesn't need any help—in fact, he is competitive with the computer. However, since these are areas which he enjoys he is willing to experiment with computer assistance providing he is clearly the boss and sets the pace. He is highly critical of the system design and is intolerant of frustration. Because he builds up his design on consecutive inputs—turnaround time, machine response must be fast in order to be useful. This usually works best in interactive systems. Examples of this area are the various computer-aided design programs. They can range from a printout of the dimensions to a design solution to interactive design in three-dimensions with color and perspective, time sequenced.

Drawing is a relevant step in the architect's strategy for evolving a design solution because he thinks graphically. He literally "sees" his solution. It is his form of data compression, data manipulation and data transmission. That is why computer use for the architect must involve graphics. It can be simple tables, charts, and even simple line drawings will do. Because the architect can visualize easily, it is not necessary for an object to be colored and shaded before he finds it meaningful. Because the embellishments are not essential the architect is not likely to pay for it for himself. The trimmings are more appropriately saved for the client. But there is a great difference between designing a computer program for an architect and for his client.

SCHOOLS

The America Institute of Architects held a seminar for architectural students on The Changing Role of Architecture and invited the deans of two eastern schools. Their answers to the question "What are you trying to teach the students?"—one answered, "To dream," the other "To think." Needless to say, in these schools there is little emphasis placed on the use of computers in architecture. There are schools where students are taught to think and dream and to use computers. To visit such a place is an electrifying experience. The Architecture Machine at M.I.T., under the aegis of Nicholas Negroponti is one such place. Students are expected to investigate, explore and contribute to the new technology. These graduates unfortunately find little place for their skills in the architecture profession and often go back into the university or to other industries and so the dissemination of new techniques slowly reaches the profession.

CONCLUSION

And so . . . how is the use of computers coming along in the field of architecture? Slowly, and for good reasons.

First, it is a complex field which covers many disciplines and it is difficult to know which parts will take to computerization.

Second, the computer people have not understood the architect. They have not amplified his strengths nor have they mitigated his weaknesses. They do not know how to bring the computer to him nor how to bring him to the computer. They don't even know of what stuff his dreams are made.

Third, progress in computer technology, for the architect, has been mostly in the universities and in the larger firms. The universities are often not aware of the real-life problems of the architect. The larger firms are too few in number and have too often been content with the computer's power to impress, rather than its power to compute.

But, the smaller firms, of which there are many, are beginning to get into the picture. After all, one can now buy a computer for the cost of a sports car. Hopefully, these small firms will contribute to the use of computers because more people, who are architects, will be using them. In addition, the new areas into which the architect is heading, are eminently suited for computer assistance and indeed many of these areas have been substantially touched by computers. Under the impetus of becoming extinct, there will be architects who, with the help of computers will finally make it into the 20th century.

APPENDIX

A selected list of articles dealing with computers and architecture. Most of them have appeared in architectural journals. They are listed chronologically because it provides an historical overview of architectural interest in computers and computer related areas.

1965 Cowan, H. J., "Industrialized Architecture," *A.I.A. Journal*, pp. 51-57, April.

1966 Ludwig, M. E., "Prejudice and the Computer," *A.I.A. Journal*, p. 70, July.

1966 Stewart, C. D. and F. de Serio, "Design with Computers? It's what's Happening, Baby!," *Progressive Architecture*, pp. 156-158, July.

1967 Swinburne, H. H., "Change is the Challenge," *A.I.A. Journal*, pp. 83-90, May.

1968 Berkeley, E. P., "Computers for Design and Design for the Computer," *Forum*, pp. 60-65, March.

1969 Catalano, E., "A Case for Systems," *Progressive Architecture*, pp. 162-166, May.

1969 Farrell, P. B., Jr., "How Computerized Land Development

Program Aids Architects, Contractors and Owner/Developers," *Building Construction*, pp. 68-69, September.

1969 Teicholz, E. D., "Architecture and the Computer," *Forum*, pp. 58-61, September.

1970 Katz, Genevieve and Lou Katz, "Capturing the Third Dimension," *Computer Decisions*, pp. 50-53, October.

1970 Milne, M., "From Pencil Points to Computer Graphics," *Progressive Architecture*, pp. 168-176, June.

1971 Bazjanac, V., "Computer Simulation: A Realistic Assessment," *Progressive Architecture*, pp. 84-87, June.

1971 Horsley, C. B., "Architects find Computer a Friend," *New York Times*, p. 6, August 29.

1971 Miller, W. R., "Computers in Architecture," *Datamation*, pp. 20-26, September 15.

1972 Hanon, J., "Designing Buildings by Computer," *New Scientist*, pp. 429-432, August 31.

1973 Stewart, C. D. and K. Lee, "Can a 54-year Old Architectural Firm Find Romance and Happiness with an Interactive Computer System?," *Progressive Architecture*, pp. 64-72, July.

1975 Berger, H., "The Engineering Discipline of Tent Structure," *Architecture Record*, pp. 81-88, February.

1975 Eastman, C. E., "The Use of Computers Instead of Drawings In Building Design," *A.I.A. Journal*, pp. 46-50, March.

SYSTEMS

Computer Systems
Systems Management
Networking
Business and Industry Systems

Prospective capabilities in hardware*

by MARGARET K. BUTLER
Argonne National Laboratory
Argonne, Illinois

ABSTRACT

Today we can look back over the past thirty years and view the entire history of the electronic digital computer! In addressing the topic of prospective capabilities in hardware, this paper first attempts to extrapolate, from today's state-of-the-art and vantage point, general industry-wide trends and likely achievements. Then, an effort is made to cover in more detail specific areas of interest to the ERDA community. Subjects discussed include available large-scale computers—their architecture and viability. The microprocessor's impact on computer systems is considered, and potential applications of the microcomputer are identified. Then mass storage offerings and their role in predicted memory hierarchies is assessed; progress in the development of alternate storage technologies is reviewed. New products and anticipated innovations in peripheral and input-output equipment, probably the most lethargic segment of the dynamic hardware market, are described, and a survey of network and communications activity examines future directions this rapidly-expanding field might be expected to follow.

Whenever possible in each of these areas, examples with descriptive characteristics, accompanied by cost and performance statistics, are presented in support of the initially-forecast broad technological trends. These examples, chosen for illustration, represent new hardware products, advanced technologies in the developmental stages, or planned enhancements of existing product lines.

INTRODUCTION

Beginning in 1946 with the dedication of the ENIAC at the Moore School in Philadelphia, this history is generally perceived as a sequence of eras. These eras, traditionally referred to as generations, are characterized by the technology employed by the computer industry during that period. The history of the computer industry, as customarily represented, is shown below with the fourth generation introduced to reflect the technological advances of this decade.

The transition from the first to second generation is clear-cut, defined by the industry's shift from vacuum tube to transistor technology. Emergence of later generations becomes blurred as the increased investment in the existing technology, coupled with the need for acceptance on a cost-performance basis, serves to retard the new alternative technologies. For example, the incorporation of large semiconductor memories was postponed by economic considerations until production quantities reached proportions permitting the technology to compete with the entrenched magnetic core storage.

In addressing the topic of prospective capabilities in hardware, this paper first attempts to extrapolate, from today's state-of-the-art and vantage point, general industry-wide trends and likely achievements. Then, an effort is made to cover in more detail specific areas of interest to the ERDA community and the agency's environmental science and analysis programs. Subjects discussed include available large-scale computers—their architecture and viability. The microprocessor's impact on computer systems is considered,

	1946 . . .	1959 . . .	1965 . . .	1971 . . .
TECHNOLOGY:	vacuum tube	transistor	integrated circuit	large-scale integration
MEMORY:	mercury delay line or electrostatic	magnetic core	magnetic core or plated wire	semiconductor

Figure 1—History of the Computer Industry—Four Generations

* Work performed under the auspices of the U.S. Energy Research and Development Administration for ERDA Conference on Computer Support of Environmental Science and Analysis, Albuquerque, NM, 1975.

and potential applications of the microcomputer are identified. Then, the recently-announced mass storage offerings and their role in predicted memory hierarchies is assessed; progress in the development of alternate storage technologies is reviewed. New products and anticipated innovations in peripheral and input-output equipment, probably the most lethargic segment of the dynamic hardware market, are described, and a survey of network and communications activity examines future directions this rapidly-expanding field might be expected to follow.

Whenever possible in each of these areas, examples with descriptive characteristics, accompanied by cost and performance statistics, are presented in support of the initially-forecast broad technological trends. These examples, chosen for illustration, represent new hardware products, advanced technologies in the developmental stages, or planned enhancements of existing product lines.

TECHNOLOGICAL TRENDS

In 1970 the largest second-order suppliers to the computer industry, the semiconductor manufacturers, were doing a 348 million-dollar business annually, producing logic components and peripheral system elements while developing the technology to permit integration of larger logical functions on a single semiconductor chip.[1] Large-scale integration, the hallmark of the fourth generation, is the name given the technology used to produce high-density electronic circuits. Although not defined precisely, it is generally interpreted to imply over a hundred "gates", or individual circuit functions, at a density exceeding 50,000 components per square inch.[2]

Major benefits and industry-wide trends resulting directly from the steady improvement in production of LSI modules and fabrication techniques are:

- *reduced cost per logic function, or memory bit, with an accompanying decrease in physical size,*
- *increased complexity with implied enhanced capability and performance, and*
- *improved reliability.*

These trends can be expected to continue. Today, chips less than a quarter of an inch on an edge incorporate well over 20,000 components at a cost under a small fraction of a cent per component.[3] This low cost is achieved primarily by economies of volume production, although advances in semiconductor fabrication techniques, and adoption of computer-aided design (CAD), manufacturing (CAM), and automated component testing procedures have played a part. The reduction in the number of interconnections and components brought about by LSI has contributed to the realization of both increased complexity and higher reliability in the hardware product. It is, and will remain, expensive to produce custom-tailored LSI; standardized circuitry is required for low prices. A plot showing the density and cost of integrated-circuit components over the 1960-1980 time period reconstructed from an August 1973 Scientific American article is shown as Figure 2.[4]

Semiconductor manufacturers concentrated much of their early LSI effort on the production of computer memory arrays because of their inherent regular structure and potential volume market as a replacement for magnetic-core storage. Fabrication technology known as MOS, for metal-oxide-semiconductor, was introduced to achieve higher-component density than realizable with the older, higher-speed "bipolar" technique. Early utilization of semiconductor memories was limited, because of cost considerations, to read-only memory (ROM) control storage or to small arrays of read/write memory (RAM) employed, in hierarchical memory organizations, as buffers or caches. In 1971 IBM delivered the first System 370/145 with its semiconductor main memory, and finally, during the 1972-73 time-period, semiconductor storage overcame the cost advantage maintained for so long by magnetic-core technology.

The MOS process was also utilized in the manufacture of chips for the popular, electronic desk and pocket calculators, which created a new large-volume semiconductor market. As an outgrowth of this effort, and in an attempt to stimulate sales of its semiconductor memory modules, Intel Corporation in 1971 announced the first, programmable, single-chip LSI processor. Soon other microprocessors appeared; these "micro" versions of the traditional CPU—the computer's control and arithmetic-and-logic units—were quickly incorporated in a variety of applications ranging from electronic games to point-of-sale and bank terminals, and laboratory instrumentation. Combined

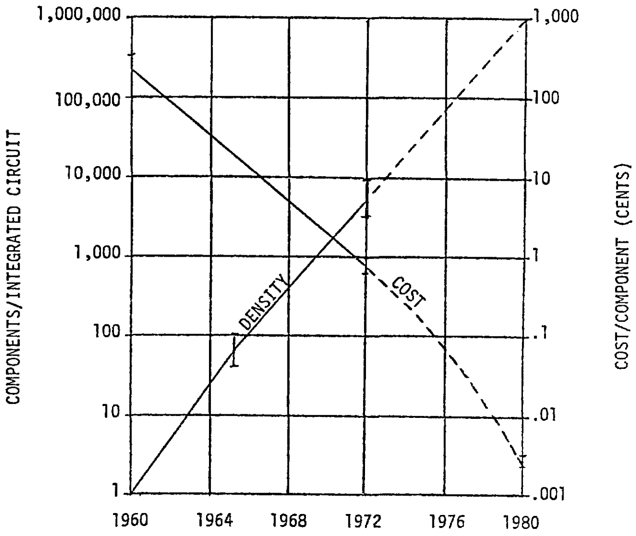


Figure 2—Density and cost of integrated circuits for the period 1960-1980

with timing, memory, and input-output facilities the long-heralded "computer-on-a-chip", or perhaps a few chips, had arrived. The microcomputer is here, and in time we can expect a nanoprocessor, capable of instruction times in the range of 100 nanoseconds. Nanoprocessors can be implemented in bipolar technology if kept simple; bipolar has yielded speeds well under 100 nanoseconds for 1024-bit RAM and ROM chips, and cost is decreasing rapidly to levels comparable to the MOS devices.[5] A silicon-on-sapphire (SOS) approach offering increased speed and component density shows promise for the future.

Entire new areas of applications have become accessible to the computer industry with the advent of the inexpensive yet powerful microprocessor. Trends in progress, or anticipated, as a result include:[1,6,7]

- *"intelligence" added to practically every type of control and data entry equipment (i.e. remote sensor and monitoring devices, automobiles, appliances, process control, data entry, graphics, and word-processing terminals),*
- *dedicated small digital systems incorporated in computer subsystems (e.g. computer peripherals controllers, communications controllers), and*
- *evolutionary changes in system design occasioned by the availability of near "zero" cost hardware and the attendant distributed intelligence possibilities.*

In addition to the trends forecast as directly attributable to LSI technology and the availability of microprocessors, a review of characteristics, components, and the organization postulated for next decade's computer indicates:[8,9]

- *processor speeds greater than 100 MIPS (million instructions per second),*
- *increased size (up to 100 megabytes) of semiconductor main memory,*
- *available on-line archival mass storage facilities using different technologies,*
- *architecture directed toward multiprocessor network configurations utilizing multilevel memory, and computer, hierarchical organizations,*
- *extensive use of microcode to accommodate dedicated or special processes and distributed-function concepts,*
- *decentralization of communications, input-output, and peripheral file management subsystems,*
- *firmware implementation of many of today's operating system features and other system software,*
- *incorporated performance-measurement monitoring, maintenance, and error-logging, and fail-soft and fault-tolerant design for increased hardware reliability and availability.*

Historically, computers have been classified as small, medium, or large-scale primarily on the basis of price and performance. Size defined the minicomputer of the sixties and the microcomputer in this decade. Such distinctions are no longer relevant; a full spectrum of computer power from microprocessor to mainframe is available for the choosing. The user pays his money and takes his choice, and he has progressively, year by year, been offered more performance for his dollar. This trend can be expected to continue, particularly at the low end of the spectrum. Tomorrow's moderately-priced computer system will afford the user much the same computing power as today's high-priced spread; the low-priced system can be expected to provide moderate capability at a reduced price, etc. Price in the computer industry is negatively correlated with quantity, or market volume. This coupling could cause the most significant impact on the scientific community, and ERDA, of all these trends. The larger, more powerful, state-of-the-art processors required for ERDA's programs will tend to represent an increasingly smaller fraction of the market; consequently, fewer such computers will be developed, and those will tend to be expensive when compared, on a cost-performance basis, to the models produced in response to market demand. Two possible reactions, should such a situation come to pass, are the approach taken by the Controlled Thermonuclear Research (CTR) Division in providing computer capability for its program, and the action taken earlier by the AEC in ensuring adequate computational resources to meet laboratory requirements. The CTR approach is to provide the top-of-the-line, number-crunching capability at a single-site, the Lawrence Livermore Laboratory, together with the network and station facilities necessary to make this power accessible to off-site program participants. The LARC and STRETCH projects were cooperative ventures undertaken by the AEC with Remington Rand and IBM to develop the computer capability needed for the Commission's research and development activities.

LARGE-SCALE COMPUTERS

About four years ago at the Idaho Falls topical meeting of the American Nuclear Society's Mathematics and Computation Division, Jack Worlton of the Los Alamos laboratory, presented an entertaining talk on a theme similar to the one covered by this paper. At that time, he noted that the IBM 360/195 and the CDC 7600, both capable of executing ten million instructions per second, were the fastest computers installed, and speculated that even faster computers would be delivered in the next few years.[10]

The number of faster computers delivered in the intervening years has been dismally small, and their power has not been uniformly impressive. ILLIAC IV, the Texas Instruments ASC (Advanced Scientific Computer), and the Control Data Corporation's STAR-100, all unconventional machines from a system architect's point of view, have arrived on the scene, each culminating about ten years of research and development effort. Standing in the wings is CRAY-1, the initial product of Seymour Cray's Cray Research, Inc., which is expected to provide five times the performance of the CDC 7600 when delivered in January of next year.

Meanwhile, work on the CDC 8600 expected as a follow-on to the CDC 7600 has been discontinued; neither the postulated IBM 370/178 nor FS, the giant's Future System has emerged. Instead, CDC is upgrading their 7600 system, IBM has announced enhancements to their 370/158 and 168, UNIVAC has souped-up its 1100 series, and Amdahl recently delivered its first machine, the 470 V/6 aimed at the IBM 370/168 marketplace. The future of large-scale, fast and powerful scientific computers is unclear.

ILLIAC IV is generally described as a parallel computer, or array processor. The concept was first explored in the early sixties at the Westinghouse Electric Corporation on the premise that large-scale computing could be characterized as repetitive execution of the same algorithm on different data streams.[11] The assumption that much of the cost of that day's conventional computer was associated with the control logic also had an influence on the design. Inherent in the Westinghouse SOLOMON design is the concept of many simple, identical processors, each programmed by a common central control unit to directly simulate the physical process being studied.[12,13] The basic system considered, consisted of multiple processing elements, (PEs), configured in an array, with each PE a complete arithmetic-and-logic unit capable of executing a full instruction set. Each element contained its own memory unit, could optionally execute or ignore a given instruction, and could transfer data to any of its four nearest-neighbors.

During the latter part of 1966 a project was initiated at the University of Illinois, funded by the Advanced Research Projects Agency (ARPA), providing for construction by the Burroughs Corporation of the ILLIAC IV parallel computer based on the prototype SOLOMON studies.[14] Sixty-four PEs were implemented in ILLIAC IV, each with a memory unit capable of storing 2048 64-bit data entities in a variety of 8-, 32-, or 64-bit fixed and floating-point formats.[15] In addition to this 8.4 million bits of main memory, the machine has a one-billion-bit diskfile secondary memory, and a one-trillion-bit archival storage subsystem. A front-end Burroughs 6500 computer was incorporated in the design to control the laser-beam read and record third-level memory, the usual array of peripheral card, printer, tape, disk and display equipment, and telephone-line communications. The machine was designed to be accessible to ARPA research contractors via the ARPANET resource-sharing network. Since the ILLIAC IV project was moved to the NASA Ames installation in California little has been heard of it, except for occasional mention of the fact that the system is not operational.[16] At the IEEE Lake Arrowhead Workshop in 1973 D. L. Slotnick, who was responsible for the project at the University of Illinois, was quoted as commenting that in an up-to-date miniaturized ILLIAC IV the processing elements would be in the reading heads of a disk.[17]

Starting with the overlapping of input-output and peripheral device operation with CPU execution 15 years ago, designers have successively introduced forms of parallelism in the initially serial stored-program computer in an attempt to achieve higher-performance with existing components and state-of-the-art technology. Pipelining techniques are implemented which segment the various stages of instruction execution and pass operands from one to the next to allow many operations to be in progress simultaneously. Multiple functional units are incorporated to permit several operations to be executed simultaneously, and microprogram control is utilized to meet the associated timing constraints. To provide the supply of operands required at a rate consonant with the CPU operation memory modules are interleaved, data paths widened, instruction stacks with "look ahead" logic and cache memories are added. In three of the available large-scale systems—the CDC STAR-100, TI's ASC, and the CRAY-1—vector capabilities have been included in the machines' instruction sets to achieve increased processor speed, leading these computers to be referred to as vector processors.

Development programs for the STAR-100 and ASC were initiated in the mid-sixties and, to date, two STARs and six ASCs have been delivered. Both STAR-100s are at the Lawrence Livermore Laboratory where personnel have been involved in the STAR development from its inception. Three ASCs are in use at Texas Instruments, one for system development and two on contract seismic applications; a fourth is employed on seismic work in the TI Amstelveen, Holland facility, and the fifth and sixth are at the Army Ballistic Missile Defense Agency Huntsville research center and the NOAA Geophysical Fluid Dynamics Laboratory at Princeton University. When Cray Research, Inc. was established in April of 1972 work began on the initial CRAY-1 machine. It is expected to be available for delivery in January 1976, and a second unit is under construction in Chippewa Falls.

A table summarizing characteristics of these three machines is shown as Table I.

Several features of these machines deserve mention as illustrative of the trends projected earlier. In the STAR-100 the CPU's stream unit, which directs the instruction and operand streams into the arithmetic unit, uses microcode resident in two 80-nanosecond ROM components to initiate and terminate vector and string operations and to monitor interrupt conditions.[18] When an interrupt occurs, information necessary to restart is saved, an interrupt flag is set, and the microcode program triggers the exchange from job to monitor mode. In the stream unit, too, is the 256 64-bit

TABLE I—Characteristics of the Available Vector Processors

CPU:	STAR	ASC	CRAY-1
Instruction size (bits)	32 or 64	32	16 or 32
Clock period (nsec)	40	60	12.5
Instruction stack/buffers	32 words (2048 bits)	16 or 32 words (512 or 1024 bits)	256 parcels (4096 bits)
Functional units	3 1 string unit 2 floating-point units	1, 2 or 4 pipes, with memory buffer unit & arithmetic unit	12 3 integer add 1 integer multiply 2 shift 1 pop. count 2 logical 1 floating add 1 floating multiply 1 reciprocal approx.
Program-addressable registers	256 64-bit	48 or 96 32-bit	8 64-element 64-bit 73 64-bit 72 24-bit 1 7-bit
MEMORY:			
Technology	magnetic core	bipolar semiconductor	bipolar semiconductor
Word length (bits)	64	32	64
Address space (words)	4×10^{12}	16M	4M
Data path width (bits)	512 (=1 s(uper)word)	256 (=1 octet)	64
Cycle time	1.28 μs	160 nsec	50 nsec
Size (words)	512K or 1M	128K to 16M*	1M
Organization/interleave	32 banks	8 module	16 banks
Maximum band width (words/sec)	200×10^6	400×10^6	80×10^6
Maximum band width (bits/sec)	12.8×10^9	12.8×10^9	5.1×10^9
Error checking	2 parity bits/word	single-bit error correction double-bit error detection	1 parity bit/word
LOGIC:**	TCS	ECL	ECL

* optional MOS semiconductor memory extension of up to 1M words with 1 μs cycle time and 64×10^6 band width

** TCS (transistor current switch), ECL (emitter-coupled logic)

register file which is implemented in read-or-write semiconductor hardware with a 20-nanosecond cycle time. The register file provides instruction and operand addressing, indexing, stores constants and field-length counts, and is the source and destination for the register-to-register three-address type instructions.[19] Distributed-function architecture is realized in the I/O Station concept of the STAR system. An I/O Station consists of a station control unit (SCU) and a station buffer unit (SBU) together with the hardware for the particular function being controlled. The SCU is a 16-bit minicomputer with 8-64K words of 1.2 ms core, CRT display, and refresh microdrum; the SBU is a 32K 16-bit word core memory. System autoload, diagnostic testing, and performance monitoring take place in the Maintenance Control Unit on one of the system's I/O channels; others accommodate the STAR paging, unit record, tape, and disk stations.

In the ASC system the peripheral processor assumes the system control and data management functions. It is designed as a multiprocessor with eight independent virtual processors, (VPs), sharing a common 4K 32-bit ROM, arithmetic unit, instruction processing unit, central memory, and access to the communication register (CR) file where control and status information necessary for system coordination is stored. Use of these shared facilities is distributed either dynamically, to suit individual processing demands, or equally with two 65-nanosecond cycles assigned each VP every 1.4 μs. The ASC operating system executes in the peripheral processor taking advantage of its hardware features. Most important of these, complete access to central memory, permits a single re-entrant copy of code to be accessed by all VPs.[20] ASC data communications are controlled by a TI980A minicomputer specially-equipped for the task; transfer rates up to 240,000 bits per second are supported.

The CRAY-1 design, which retains much of the flavor of the 7600, is based on a 1024-bit, bipolar RAM chip with a cycle time of 50 nanoseconds. The system consists of the central processor with 1M words of central memory, 12 I/O channels, a maintenance control unit, and a CDC 819 disk subsystem. All units are tightly synchronous with a clock period of 12.5 nanoseconds. The memory is organized as 16 banks of 65,536 64-bit words. Each memory module consisting of 64 chips, 32 per side, represents a bit position; 64 modules constitute a bank of 65,536 words. Two banks are packaged vertically per chassis with the eight memory chassis positioned four on each side of the CPU in the CRAY-1 main frame.

Contained in the central processor are four instruction parcel buffers (IPBs), over 150 program-addressable registers, and 12 independent algorithm or functional units. Each CRAY-1 instruction is either a one-parcel (16-bit) or two-parcel (32-bit) instruction. The 64-parcel instruction buffers are organized in 4 to 1 correspondence with the 16-memory banks so that the first four parcel positions in a buffer are always filled from bank 0, parcels 5 through 8 come from bank 1, etc. This feature, combined with the use of four 64-bit paths, in parallel, between memory and the IPBs, makes possible a transfer rate of four words per clock period. Buffers are filled in turn; whenever an instruction fetch from memory is required, the "least recently filled" IPB is used. The fetched instruction is always read in with the first parcels regardless of its memory bank source and associated buffer-position destination.

The machine complement of program-addressable registers resembles a greatly-expanded CDC 7600 collection. Registers have been added to hold vector instruction parameters and to serve as source and destination addresses for operands in, up to 64-element, vector arithmetic and logical instructions. In addition, the eight principal address-length registers used as address registers for memory reference and index registers, and the eight principal word-length registers which act as source and destination registers for scalar operands, are each backed-up by 64 secondary registers designed to provide quick-access, temporary storage. Operands can be recalled from secondary to principal register storage in one clock period, and the secondary registers can be loaded from memory, or stored away in stream-fashion with a single block copy command at the rate of one clock period per operand after a startup period. Contents of a vector register are stored in, or loaded from, memory by specifying the initial address, an increment for succeeding addresses, and the vector length, at a one-clock-period rate once eight elements have been transferred.

Functional units are designed with one-clock period segmentation; the source and destination addresses are limited, and the algorithm chosen so that the time required for each unit to complete its task is fixed.[21] In vector mode results are produced at a one-clock-period rate, and these results may interact with other vector instructions in "chained" operations since all functional units have the same result rate. Three integer add units, an integer multiply, two shift units, a population counter, two logical operations units, and floating-point add, multiply, and reciprocal approximation units are included in the central processor. The machine performs floating-point division by reciprocal approximation; four instructions are necessary to obtain 48-bit precision. When vectors longer than 64 elements are used, programming is required to divide them into 64-element sequences for processing.

Twelve full-duplex, 64-bit wide, input-output channels are included in the CRAY-1 system; however, the only peripheral equipment supplied will be the CDC 819 disk which is used as the resident device for the operating system. The system described is priced at 7 to 8 million dollars and purchasers are expected to acquire their unit record, disk, tape, display, and terminal equipment from other vendors.

Today, although both STAR and ASC systems are in operation it is difficult to find performance comparisons with conventional machines or published benchmark problem results. The "promised" computing power of vector processors a decade ago has yet to be realized. The architecture is designed to optimize vector-mode operations in which the vector elements are at contiguous storage locations, and the number of elements in the vector is large; the latter is especially true in the STAR-100 implementation. Consequently, it is essential that programs written for these vector processors exhibit a high ratio of vector to non-vector (scalar, branch, test, execute) instructions and that both code and data be ordered to minimize memory references and maximize parallel operations. Over the past six years LLL computer personnel have been working on the construction of the STAR operating system and related software to achieve their announced goal of incorporating the machines as worker computers in the Laboratory's OCTOPUS time-sharing network.[22] During this same period they have developed techniques and methods for transforming existing large production codes into "vectorized" STAR code.[23] This has proved to be an arduous task.[24]

Texas Instruments has, since 1968, been developing a sophisticated FORTRAN compiler, ASC NX, with array-oriented language extensions to generate "vectorized" code for their machine.[25] The NX compiler translates FORTRAN DO-loops into vector instructions, and, if instructed that the code is for a multi-pipe machine, can introduce parallel instruction streams. An extensive optimization analysis is performed and as output, the programmer receives information intended to assist hand-tailoring of the source code to allow further iterative compiler optimization, if desired. Factors to be considered, and techniques, for improving the efficiency of execution of ASC scalar instructions were presented in a recent paper.[26] Table II summarizes performance statistics which have been reported for the two machines.

The CRAY-1 second generation vector processor is obviously intended to take care of many of the problems that have been encountered in attempting to use the first-generation processors, and both CDC and TI are planning for the future. CDC expects to replace their core memory with a bipolar semiconductor memory and to add a front-end computer to the system. Before the year is out, they intend to announce STAR as a product, send staff to Livermore to assist with the STAR-OCTOPUS merger, deliver model 103 to

NASA Langley, and add 104 to their own CYBERNET computer service network. TI is readying a 61,500 word per second mass-storage videotape subsystem for use with the serial 4 ASC at the Princeton Geophysical Fluid Dynamics Laboratory, and that machine's main memory will probably be expanded to 4M words utilizing a 1024-bit memory chip with a 100 nsec store time instead of the present 256-bit chip. Serial 7 will be delivered to the Naval Research Laboratory in January of next year and late in 1976 announcement of a 6-megaword compatible channel can be expected. Presently being looked at as a cost reduction measure is a 16M-word-memory design incorporating a 4K MOS chip with 200 nsec access and increased interleaving. The company is interested, also, in locating a customer seeking faster scalar capability; they believe a two- to fourfold increase in speed can be realized within two years, following such an order.

While the array and vector processors were being designed and constructed, the mainstream large-scale computer systems were evolving toward a flexible, modular architecture in which the major system components such as memory, processor, and input-output control are treated as logically separate entities. Keenly aware of the user's evergrowing investment in applications programs and computer-based data, manufacturers have attempted to structure their product lines to accommodate the user. The number and performance capabilities of the component units can be tailored to meet budget constraints and growth requirements, and technological advances offering improved component performance can be incorporated quickly and effectively, all with minimal impact on the user's operation. Increasing emphasis is being placed on system reliability and availability.

Control Data Corporation's 7600 has found wide acceptance in ERDA laboratories and the nuclear industry where 18 of the 38 delivered systems are employed. Later this year the company plans to enhance this system with expanded and improved memory components. Characteristics of the current and proposed 7600 SCM and LCM units appear in Table III. The CTR Computer Center at Livermore is scheduled to receive the first of the small semiconductor memory models during the latter half of this year, and the revamped 7600 can be expected to appear in CDC's product line about a year later.

TABLE II—Performance Statistics and Comparisons of Vector Processors Relative to the CDC 7600 and IBM Model 195

COMPUTATIONAL MODEL OR PROBLEM	CDC 7600 OR IBM 360/195	STAR	ASC	CRAY-1
Scalar	1	0.25		5
Vector				
(10 elements)	1	0.4		6
(25 elements)	1	1		
(200 elements)	1	5		
(1000 elements)	1	5		10
Scatter-Gather	1	0.67		5
1-D HYDRO	1	1.6		
Matrix Inversion				
(25×25)	1		1.58	
(50×50)	1		1.75	
Matrix Multiplication				
(25×25)	1		1.27	
(50×50)	1		1.32	
GFDL models	1		3 to 14	
GFDL jobstream	1		2.5 to 3	

IBM announced the 168-3, an enhancement of their 370 Model 168, in March of this year; first customer shipment is scheduled for June.[27] Using a new 1K bipolar chip IBM has increased the machine's 80 ns cache buffer memory capacity from 16K to 32K bytes. New microcode in the processor's reloadable control increases the speed of a number of frequently-used instructions and internal interrupts. A service processor added to the 168-3 monitors and stores machine status information to aid engineers in servicing and diagnosis of machine failure; the service processor also provides a teleprocessing interface with the company's Field Engineering Large Systems Support Center.[28] A 5 to 13 percent increase in performance has been predicted for the improved 168, which is priced at 5 million dollars when equipped with the top-of-the-line 8-megabyte memory.

TABLE III—Characteristics of CDC 7600 Memories

	CURRENT		PROPOSED	
	SMALL	LARGE	SMALL	LARGE
Technology	core	core	semiconductor	core
Size (words)	32K or 64K	256K or 512K	64K or 128K	512K, 1M, or 2M
Organization (banks)	16 or 32	4 or 8	16 or 32	2, 4, or 8
Cycle time (nsec)	275	1760	110 read, 165 write	1760
Holding register (words)		8		16
Error checking	5 parity bits/word	4 parity bits/word	single-bit error correction double-bit error detection	

In March, too, Sperry Univac announced its 1100/40 system, successor to the 1100 as the largest of the firm's large-scale systems.[29] The 1100/40, like the 1110, has two levels of memory: a primary memory with from 28K to 512K 36-bit words and an extended memory with 128K to 1M-word capacity. Implemented in 1K bipolar the 1100/40 main memory offers double the primary storage available with the earlier plated-wire technology. This memory has 280 ns read and 380 ns write time, while the extended MOS memory for the 1100/40 has an 800 ns cycle time. The processor component in the system is the Command/Arithmetic Unit (CAU) with 300-nanosecond basic instruction time and 1.8 MIPS capability.[30] Input/Output Access Units (IOAUs) control communications among system components and peripheral input-output. Multiprocessor configurations as announced offer up to four each CAUs and IOAUs. Gains of 10 to 25 percent in throughput over the 1110 are expected for the new system. Recently, NASA selected this system for a potential 8-million-dollar space shuttle simulation complex at the Houston Johnson Spacecraft Center. The NASA system will have 6 CAUs and 3 IOAUs extending the 1100/40s multiprocessor capability.

Advanced LSI technology is stressed in the newest entry in the large-scale computer industry, the Amdahl 470 V/6. This machine is designed to compete for the IBM 370/168 market using slightly-modified IBM software systems and, like the CRAY-1—letting the customer select his peripherals from other vendors. Central processor circuits are custom-designed emitter-coupled logic.[31] The Amdahl LSI chips, 10 mils thick and measuring 0.154 inch square, can hold up to 100 circuits; speeds on the chip are on the order of 600 picoseconds. Chips are mounted on a ten-layer multilayer board in specially-designed multichip carriers, (MCCs), which serve as the field-replaceable unit. Fifty-one MCCs, each containing about 3000 circuits, make up the 470 V/6 CPU and channel. From 1 to 8 megabytes of directly-addressable MOS memory is offered with a 16K high-speed bipolar cache and 16 I/O channels in the basic configuration. The channels provide standard 360/370 interfaces for attachment of peripheral devices. At the system console, equipped with keyboard/CRT display, minicomputer, cassette tape reader, modem, and disk storage unit, console operation, hardware control, and system maintenance functions are carried out. The first Amdahl computer, delivered to NASA's Institute for Space Studies at Columbia University the beginning of June, was up and running in a week's time.[32] Table IV shows some reported single-job NOAA benchmark comparisons of the 470 V/6 with the IBM Model 195.

TABLE IV—Benchmark Performance Comparisons of the Amdahl 470 V/6 with the IBM Model 195

		Amdahl 470 V/6	
	IBM Model 195	CPU	Total
1A	1	.71	.69
1B	1	.58	.59
2	1	.52	.56
2 Main step	1	.52	.57
3	1	1.31	1.06
3 Main step	1	1.30	1.03

MICROPROCESSORS AND MICROCOMPUTER APPLICATIONS

In fiscal year 1974 over half of the computers in the Atomic Energy Commission were Digital Equipment Corporation machines.[33] A large majority of these were minicomputers incorporated in dedicated laboratory applications. It is reasonable to expect that, in the future, with the growing availability of microprocessors and microcomputers, the instrumentation and computer requirements for many ERDA laboratory applications will be supplied by cheaper, smaller, more specialized, and more reliable LSI technology. This year DEC introduced an LSI-11 microcomputer at the low end of its PDP-11 product line. An LSI-11 configured with 64K bits of read/write memory, 64K bits of PROM (programmable ROM) memory, a 16-bit parallel input-output interface, and floating-point arithmetic is available at a cost of around 1500 dollars. The PDP 11/40 instruction set is emulated in microcode.

Two laboratory applications of microcomputers at LLL were reported in a paper presented at a February computer conference.[34] One dealt with tritium monitoring in various environments; the second was concerned with calculations on spectral data output from a pulse height analyzer. LSI technology is particularly adaptable to the area of environmental monitoring, and the low cost of microprocessors and microcomputers can be expected to lead to new applications not previously considered.

MASS STORAGE SYSTEMS

In the past year with the announcements of the IBM and CDC mass storage systems there has been a resurgence of interest in archival mass storage to replace manually-mounted magnetic tape, to allow installations to service evergrowing numbers of interactive users, and to place burgeoning data bases on-line particularly in a multi-computer environment. To appeal, such systems must be available at a cost approximating that of magnetic tape and be easily integrated into the customer's operation.

The IBM 3850 Mass Storage System (MSS) is a hierarchical storage system capable of storing and managing up to 472 billion bytes of data on-line.[35,36] The data is stored on 64-foot lengths of magnetic tape, about 3 inches wide, "spooled" in plastic cartridges, 4 inches wide and 2 inches in diameter. Data recorded on the tape as IBM 3336-1 cylinder images, each cylinder appearing at a fixed location on the tape. A single cartridge can contain 202 cylinders; a pair of cartridges, the equivalent of one disk pack, is referred to as a mass storage volume (MSV). All space in the MSS is managed in terms of these MSVs. Cartridges are stored in cells in a honeycomb arrangement in the unit. Data are transferred from the cartridges to staging buffers (dedicated 3330 drives) when requested. Once staged there, data are accessible just as any other data resident on a 3330. When no longer needed, and the space on the staging device is required for other data, any "cylinder" containing new or updated data is destaged back onto the data cartridge. MSS uses a "least recently used" replacement algorithm. Important concepts in the IBM 3850 implementation are the ideas of virtual device and virtual volume. The virtual device concept allows more drives to be addressed than actually exist in the hardware configuration. The virtual volume concept allows many partial MSVs to reside on a single staging drive, or different parts of an MSV to reside on several staging drives. Industry sources believe IBM already has 700 to 1000 firm orders for the 3850.[37]

The CDC system is composed of a mass storage adapter (MSA) unit and a mass storage facility (MSF) with two, three, or four read/write stations mechanically coupled to a cartridge storage unit.[37] The cartridge file provides storage for up to 16 billion characters. The MSF may be configured modularly in sizes greater than 16 billion bytes. A single MSA can handle up to 8 or 16 elements depending on the model; an element is defined as either a read/write station or a cartridge storage unit. Up to 8 MSAs can be attached to a single 3830-2 controller. Data in the MSF are recorded on 100-inch lengths of 2.7-inch-wide magnetic tape in a 9-track format at 6250 bpi density. One hundred and forty-four tracks can be recorded. The tape strip, containing up to 8 megabytes of information, is enclosed in a plastic cartridge, measuring 1.125″ x 1.25″ x 3.3″. Two thousand such cartridges are stored in the rectangular array of cells making up the cartridge file. When a data set is required the applicable data set cartridge is located by its x-y coordinate address in the file, selected from its cell by the unit's cartridge selector, a pneumatic pick mechanism, and transported to a read/write station. At the station the cartridge is opened, the tape unwound, and drawn into two vacuum columns for reading. After being read the tape is rewound into the cartridge, and the cartridge is sealed and returned to its file location. The tape is never detached from its protective housing. Data can be staged to any available disk space in the host system, or staging can be eliminated altogether. Data sets can be read directly to main memory and returned directly to the cartridges, if desired. Two cartridge I/O drawers, with 8 cartridge slots apiece, allow the operator to enter and remove cartridges from the unit manually, and a write-inhibit plug can be used to ensure that master library tapes are not written over accidentally. A CDC MP16 minicomputer incorporated in the MSA performs on-the-fly error correction and detects illegal commands and incorrect sequences. Control Data estimates ten million dollars has been spent over the past five years developing this system.[38] One of the first systems scheduled for shipment in the fourth quarter of 1976 will go to LLL.

Two other currently-available mass storage systems are the CalComp Automated Tape Library (ATL) and the Ampex TBM Mass Storage System. The ATL was originally designed and marketed by Xytex Corporation of Boulder, Colorado.[39] In early 1974 CalComp acquired the company, which was formally merged into XTX, a wholly-owned CalComp subsidiary in January of this year. ATL is a modular system with a basic configuration of a control unit, two storage units, a reel selector mechanism, and one automatic reel-mounting unit servicing one tape drive. This configuration is capable of storing 762 tape reels, and can be expanded by incremental addition of storage units and tape drives to accommodate 6250 magnetic tape reels and 32 tape drives. Under computer control the storage system automatically retrieves requested tapes, mounts them on the self-threading tape drives for system use, and dismounts and refiles them upon job completion. A complete inventory of status, usage, and location information for all tape files is maintained.

Ampex's random access terabit memory utilizes standard 2-inch-wide video tape recorded in a block format to provide up to 350 billion bytes of data on-line at a cost of about .0001 cent per bit.[40,41] Each block is identified by a unique address allowing block-address searches, forward and backward, at 1000 inches per second. The system is in two parts: a data storage section composed of the transport modules, transport drivers, and data channels, and the control section with its storage control processor for system control and data channel processors to control data transfer between the TBM and the host computers. Each transport module includes two transports and has an 11-billion-byte capacity; a TBM system can have up to 32 transport modules, and with up to six transport drivers up to six concurrent accesses are possible. Each data channel unit performs independent read and write operations at a rate of 700 kilobytes per second. With three data channel units, the system maximum, six simultaneous read/write commands yield throughput of 4.2 megabytes per second. Switching matrices allow any transport to be accessed by any driver and data channel and provide flexibility for dynamic reconfiguration and off-line maintenance of part of the system. The TBM is capable of operating independently of host processors; when used with host computers, the system can either stage data to shared disks or route the data directly to host channels.

The availability of two additional mass storage systems is questionable at this time. These two are the Precision Instrument trillion-bit laser-based mass storage System 190, a follow-on to their UNICON system which is installed on ILLIAC IV, and the Grumman Masstape system, presently under reevaluation by its developers.[42,43] System 190s had been ordered by the Social Security Administration, and by Holifield National Laboratory for ERDA Technical Information Center applications, before the company went into receivership. Refinancing plans have recently been announced, and it is now probable System 190 will be marketed. Characteristics of these two systems, as well as those of the IVC-1000 video tape recorder being adapted by TI for the GFDL ASC, have been included in the Table V mass storage system summary.

ALTERNATE STORAGE TECHNOLOGIES

Alternate storage technologies bridging the classical memory access gap between main memory and the peripheral storage devices, and at the same time, offering the per bit cost of the rotating disks and drums, have been the object of continuing research. During the past five years three areas in which there have been significant development efforts are: charge-coupled devices (CCD), bubble domain technology, and electron-beam addressed memories. The first two have been described as belonging to the "moving the bits to the sensor" philosophy, while the third applies the "moving the sensor to the bits" approach.[45]

The CCD is, basically, a shift register for analog signals made in the form of a string of MOS capacitors. CCD memories, being serial in nature, are block- rather than bit-oriented. Primary incentive for CCD memory development is its potential low cost derived from higher packing density and the inherently simpler fabrication. One drawback is the long access time associated with the serial nature of CCD technology. At this year's National Computer Conference, Fairchild described their CCD 450 9216-bit storage which is organized as 1024 9-bit bytes.[46] The unit is viewed as a possible storage unit for portable terminals where its byte format and low power features can be exploited.

Throughout the past five years magnetic bubble memories have been envisioned as a technology which is just two to three years away. Their most attractive characteristic is their non-volatility. Like CCD technology, the bubble domain systems offer low power consumption and high density per chip. By November of last year prototype bubble memory systems had been constructed by three companies, and Rockwell presented three illustrations of the technology at the NCC.[47] One application was as a spacecraft tape recorder replacement, the second as an alternative to a tape cassette or floppy disk in a data logging device, and the third example was a block-organized memory for a data processing system. Access time quoted for the 64K-1024K, 16-bit, block-organized memory was 0.5 to 1 ms at a 0.05 cent-per-bit price.

Electron beam technology which dates back to first-generation computer memories, is once again showing promise. General Electric's Research and Development Center is presently building 32-million-bit BEAMOS (Beam-Addressed MOS) memory modules in pilot quantities.[48] The BEAMOS module consists of a memory plane and electron-beam accessing system enclosed in a glass envelope; it has an access time of 30 μs and a transfer rate of 10 megabits per second. A "matrix electron lens" with 289 lenslets is used to direct the cathode-ray-tube beam to read, write, or erase at precise sites on the four silicon storage chips of the memory plane. Each of the chips holds 8 million bits. In a multimodule system, 16 or more tubes can be linked to provide a 512-million-bit or greater storage capacity. Data transfer rates of 160 megabits per second could be realized by accessing a 16-module system in parallel. Cost of the electron-beam addressed memory system is estimated to be in the .02 to .1 cent/bit range.

PERIPHERAL AND INPUT-OUTPUT EQUIPMENT

Although the industry has been actively seeking an all-electronic auxiliary memory, this has not seemed to discourage efforts in disk technology. New products introduced include the Storage Technology 8800 "super disk" with 800 megabyte capacity and the CDC 819 high-capacity disk subsystem with a 412-million-character capacity. Delivery of the 819, already incorporated in the STAR systems, is scheduled for August. This disk has an average access time of 50 ms, 8.3 ms average latency, and a transfer rate of 6.2 million characters per second. IBM and the IBM-plug-compatible manufacturers continue to turn out the 3330 200-megabyte disks and the Winchester 3340 with its 25 ms average access time. Electronic News reported in May that disk drive shipments by U.S. firms were expected to reach 1.2 billion dollars this year, and tape drive shipments 1.3 billion. The ANSI 6250 magnetic tape standard will soon be out for public review and comment. This tape will become and remain a high-volume product due to its high data rate, low cost, and interchange capability.

Three I/O product areas projected for growth are high-speed printers, intelligent terminals, and floppy disks. Both Honeywell and IBM have introduced non-impact, high-speed printers. Livermore has had two of the Honeywell printer subsystems, which use Honeywell 716 minicomputers as controllers, in use off-line for over a year. Each 12,000-18,000 lpm unit has a read-only memory character generator which prints up to 192 characters using an electrostatic technique. Cost of the Honeywell subsystem is $162,120. In April of this year IBM announced a printer system that combines laser and electrophotographic techniques to

TABLE V—Summary of Characteristics of Mass Storage Systems

	AMPEX TBM	CALCOMP ATL	CDC MSS	IBM MSS	PI 190	GRUMMAN MASSTAPE	IVC-1000
Host interfaces:	IBM 360/370, CDC, DEC	IBM 360/370	IBM 370 CDC	IBM 370 VS only			IBM 370[44] TI ASC
Capacity:	350B bytes	762-6250 reels	16B byte increments	35B-472B bytes	16B byte increments—128B bytes	11.8B byte increments	
Media:	2″ video tape on 10½″ reel	½″ tape on 10½″ reel	2.75″ tape in plastic cartridge	3″ tape in plastic cartridge	rhodium-coated mylar strip	½″ tape in cartridge	1″ video tape on 12½″ reel
Unit capacity:	11B bytes/ module (module=2 reels)	standard 3420 tape	8M bytes	50.4M bytes	200M bytes	36M bytes	9×10^{10} bits
Unit cost:	$150/reel		$15/cartridge	$20/cartridge	$20-$25/strip		
Recording:	transverse VTR, block-addressable	longitudinal	longitudinal	helical VTR	laser beam	file-addressable	helical VTR
Access time:	search 1000 ips	11 to 14 sec	5 sec	3 to 8 sec	7 to 10 sec track access: 220 ms	avg 5 sec max 15 sec	search 400 ips
Data rate:	4.2M bytes/s (6 channels)	1.25M bytes/sec	806K bytes/s from disk 400K bytes/s staging	806K bytes/s from disk 200K bytes/s staging	440K bytes/s	100K bytes/s single file 350K bytes/s per host	1M bytes/s
Price:	$400,000 to $1.5 million	$70,000 to $250,000	~$400,000 (16B system)	$700,000 to $4.5 million	~$400,000 (16B system)		

print up to 13,360 lpm on plain paper. It may be purchased for $310,000, and first customer shipments are scheduled for the third quarter of 1976. The 3800 can be attached to any channel and can print up to 225 characters, in four character sets at a time, on any of 50 page sizes. Characters can be produced in 12 to the inch, and 15 to the inch, as well as in the standard 10 to the inch size. A number of standard character fonts are included. Xerox markets a 4000 lpm page printer, known as the 1200, which is available on a rental basis only; the cost is $1500/month with a monthly page charge of 11 mills each for the first 100,000 pages, 7 mills each for the next 200,000, and 4 mills for each page thereafter. UNIVAC is rumored to be working on a 14,000 lmp laser printer, and at NCC Canon distributed literature on their off-line 1000 to 4000 lpm laser beam unit. Paper costs for the equipment vary as shown in Table VI.

Reports on terminals have predicted a rise from fewer than 200,000 in 1970 to almost 3,500,000 by 1980.[49] LSI technology coupled with the growing trends to communications-based systems and distributed-function architecture make even that estimate appear conservative.

Floppy disk drive sales are expected to show a five-fold increase between 1975 and 1980. Applications of floppies are primarily in data entry systems, intelligent and point-of-sale terminals, and as minicomputer and microcomputer peripherals. The disk holds up to 1898 128-character records and is a very inexpensive memory unit; today's prices vary between $4.50 and $8.00 and reductions can be expected as market volume increases.

NETWORK AND COMMUNICATIONS ACTIVITY

The network and communications area is the area destined to receive the most attention from the computer manufacturers in the next five years. An IEEE Computer Society glossary defines a computer network as an interconnected group of independent computer systems which communicate with one another and share resources such as programs, data, hardware, and software. Under this definition most of today's computer facilities can be classified as networks. With data traffic in the United States growing at an annual rate of 35 percent, computer manufacturers have become increasingly committed to the development of

TABLE VI—A Comparison of Available High-Speed Printer Subsystems

	Honeywell	IBM	Xerox
Technology:	Electrostatic	Laser	Xerographic
Speed (lpm):	12,000-18,000	up to 13,360	4000
Paper-use fees:	$2.61-$2.31/1000	$2.30/1000	1.1 to 0.4¢/ copy
Purchase price:	$162,120	$310,000	NOT SOLD
Rental:	$3667 minimum	$7,344	$2100 or $2600

coordinated hardware and software systems for communications-based information networks.

IBM's effort to provide a single uniform environment for data communications is identified as Systems Network Architecture, or SNA.[50] Digital Equipment Corporation calls its DECNET software a set of tools to allow intersystem communication,[51] and CDC has opted for the title Network Communications System (NCS) to describe a set of hardware and software being produced to accomplish the four processing functions:

(1) interfacing one or more host computers;
(2) interfacing a great number of terminals, often of widely varied use and manufacture;
(3) routing data between its sources and its destinations;
(4) interfacing, within the host computer, between user programs and external communication equipment.[52]

The manufacturers are employing state-of-the-art technology and distributed-processing concepts in implementing communications subsystems to perform some of the functions previously assigned the central processor, such as communications management, data formatting, device control, and even application processing. In addition, they are defining the system software protocols for interfacing. SNA software elements defined include: Virtual Telecommunications Access Method (VTAM) to manage the connection and disconnection of terminals to application programs; NCP, the Network Control Program, to perform tasks associated with physical network requirements; and SDLC, the data transfer protocol defining the format for data exchange between two network nodes. NCP can be resident in a programmable communications controller.

DECNET utilizes three protocols, as well. The first, DAP, for Data Access Protocol, is designed to allow programs on one node of a network to use the I/O services of other nodes; the second, the Network Services Protocol, handles the routing of messages within or between systems; and the third, DDCMP, Digital Data Communications Message Protocol, serves as the data transfer, or link, protocol. Burroughs recently released a Burroughs Data Link Control (BDLC) designated as that company's data transfer protocol. The next step is to get all computer manufacturers to agree on common protocol. It appears adoption of an ANSI standard for data link control is about a year to a year and a half away. Until such a standard is adopted the incompatibility of communications hardware and software products will deter network growth. Once the bit-oriented data transfer level standard has been accepted, standards for the higher-level network protocols should be forthcoming.

SNA hardware to date consists of terminals and communications controllers, some general-purpose and others dedicated to a specific industry, or application. Any new IBM system line, such as FS, is expected to be communications-based. CDC's first step in NCS hardware development was their 2550 Host Communications Processor (HCP) designed to interface terminals to a host computer, such as a CYBER 170. The HCP hardware includes a 16-bit microprocessor and is scheduled for delivery the latter half of this year.

In addition to the prospects of networking capabilities originating with the computer manufacturers, the Argonne, Berkeley, and Brookhaven laboratories are scheduled to complete connections to the ARPANET this year, and the first commercial packet-switched carrier is expected to begin service to customers in a seven-city operation soon.[53]

Four years ago when Jack Worlton concluded his presentation it was with the thought that the long-awaited computer revolution was still a way off. His criterion for determining the advent of such a revolution required that the computer effectively penetrate to a large fraction of the private homes in our society. At such a time he felt it would be valid to speak of a computer revolution, and not until then, and he speculated that maybe this could happen by 1984. It will not be long before the microprocessor, incorporated in washing machines, automatic dryers, etc. will have done just that! By 1984, a personal computer equipped with keyboard/CRT, hard-copy device, and floppy disk could be available for under $500. With corresponding advances in information processing network technology, it will at some point become unprofitable to collect people in office buildings simply to talk to each other and pass papers. Given suitable computing and communications media, work of this nature can be done, possibly more effectively, on a distributed basis by people working at home. This "cottage industry" concept has some interesting social consequences, not the least of these is a reduction in the consumption of energy, which is in short supply, through increased information processing capability, which appears to have no perceivable limit. Indeed, the very existence of the human species is proof that information processing "power" does have survival value!

ACKNOWLEDGMENTS

In preparing this paper information and assistance were requested from a number of sources, who contributed written material and presentations of ongoing research and development efforts. The assistance and cooperation of the following individuals, associated with the computer manufacturers listed, is gratefully recognized:

W. Andreen, D. Baugatz, P. Crum, R. H. Lee, G. Van Guilder, and D. M. Wareham	Control Data Corporation
L. Davis, G. Grenander, and G. Hanson	Cray Research

G. Bell, H. Bynum, P. Christe, R. Corbin, D. Reinke, R. Rutledge, and R. B. Wham	Digital Equipment Corporation
E. E. Hayes	IBM
D. Best, G. Boswell, A. Riccomi, D. Sifferd, and D. Wedel	Texas Instruments
P. Dodge, H. Gyllstrom, F. Lexa, and J. Macina	UNIVAC

At Argonne the secretarial services and editorial help provided by Catherine Hughes, technical discussions with colleagues D. Jacobsohn and W. Lidinsky, and the cooperation of R. Royston are acknowledged with thanks.

REFERENCES

1. Nelson, James C., "The Economic Applications of Microprocessors on Future Computer Technology and Systems," *AFIPS Conference Proceedings*, May 19-22, 1975, Anaheim, p. 629.
2. Heath, F. G., "Large-Scale Integration in Electronics," *Scientific American*, 222, 2, February 1970, p. 22.
3. Vacroux, Andre G., "Microcomputers," *Scientific American*, 232, 5, May 1975, p. 32.
4. Hittinger, William C., "Metal-Oxide-Semiconductor Technology," *Scientific American*, 229, 2, August 1973, p. 48.
5. Laliotis, Theodore A., "Microprocessors Present and Future," *COMPUTER*, 7, 7, July 1974, p. 20.
6. Wickham, Robert F., "The Microprocessor Market—Present and Future," *1974 WESCON Technical Papers*, 18, 11/1.
7. Saba, Mona M. and Jack D. Grimes, "Microprocessors: A Component for All Seasons," *1974 WESCON Technical Papers*, 18, 23/3.
8. Withington, Frederick G., "Beyond 1984: A Technology Forecast," *Datamation*, 21, 1, January 1975, p. 54.
9. Joseph, Earl C., "Storage System Architecture: Future Trends and Technology Implications," *1974 IEEE Intercon Technical Papers*, 23/1.
10. Worlton, William J., "Future Prospects in Computer Technology," in *Proc. of Conf. on New Developments in Reactor Mathematics and Applications*, March 29-31, 1971, Idaho Falls, CONF-710302, Vol. 2, p. 1086.
11. Murtha, John C., "Highly Parallel Information Processing Systems," *Advances in Computers*, 7, 1966, p. 1.
12. Slotnick, Daniel L., W. Carl Borck and Robert C. McReynolds, "The SOLOMON Computer," *Proc. Fall Joint Computer Conference* 1962, p. 97.
13. Gregory, J. and R. McReynolds, "The SOLOMON Computer," *IEEE Trans. on Electronic Computers*, EC-12, December 1963, p. 774.
14. Barnes, George H., et al., "The ILLIAC IV Computer," *IEEE Trans. on Computers*, C-17, 8, p. 746.
15. Slotnick, D. L., "The Fastest Computer," *Scientific American*, 224, 2, February 1971, p. 76.
16. *Computation Department Annual Report—July 1973 to June 1974*, UCRL-50023-74, September 1, 1974.
17. Poppelbaum, W. J., "Information Hardware . . . Again," *COMPUTER*, 7, 3, March 1974, p. 36.
18. *Control Data STAR-100 Computer, Hardware Reference Manual*, Control Data Publication No. 60256000-08, December 15, 1974.
19. Purcell, Charles J., "The Control Data STAR-100—Performance Measurements," *AFIPS Conference Proceedings*, 43, 1974, p. 385.
20. Watson, W. J. and H. M. Carr, "Operational Experiences with the TI Advanced Scientific Computer," *AFIPS Conference Proceedings*, 43, 1974, p. 389.
21. *The CRAY-1 Computer*, Preliminary Reference Manual, June 1975.
22. Requa, Joseph E., "STAR: A System Programmer's View," *Proc. IEEE Computer Society Int'l. Conf. COMPCON72*, September 12-14, 1972, San Francisco, p. 13.
23. Kransky, Valere J., E. Dick Giroux and Gary A. Long, "Parallel Implementation of a Two-Dimensional Model," *Proc. of the 1973 Sagamore Computer Conf. on Parallel Processing*, IEEE-ACM-Syracuse University, August 22-24, 1973, p. 69.
24. Kishi, Tad and Tim Rudy, "STAR TREK," *Proc. IEEE Computer Society Int'l. Conf. COMPCON75*, February 25-27, 1975, San Francisco, p. 185.
25. Wedel, Dorothy, "FORTRAN for the TEXAS INSTRUMENTS ASC SYSTEM," *Programming Languages and Compilers for Vector and Parallel Machines*, Goddard Inst. of Space Studies, March 18-19, 1975, New York.
26. Gaulding, Scott N. and David P. Madison, Jr., "Optimization of Scalar Instructions for the Advanced Scientific Computer," *Proc. IEEE Computer Society Int'l. Conf. COMPCON75*, February 25-27, 1975, San Francisco, p. 189.
27. "Three VS Systems Enhanced by IBM," *Electronic News*, March 31, 1975, p. 20.
28. *IBM System/370 Model 168-3*, IBM Publication G520-2619-1, March 1975.
29. "Two New Univac 1100s Feature MOS Memories," *Electronic News*, March 24, 1975, p. 20.
30. *SPERRY UNIVAC 1100/40 Systems Hardware*, SPERRY UNIVAC Publication UP-8216, 1975.
31. *Amdahl 470V/6 Features*, Amdahl Corporation, 1975.
32. "Data Topics," *Electronic News*, June 16, 1975, p. 44.
33. *Inventory of Automatic Data Processing Equipment in the United States Government for Fiscal Year 1974*, General Services Administration, December 1974.
34. Maples, M. D., "Microprocessors in Computing?" *Proc. IEEE Computer Society Int'l. Conf. COMPCON75*, February 25-27, 1975, San Francisco, p. 69.
35. Johnson, Clayton, "IBM 3850—Mass Storage System," *AFIPS Conference Proceedings*, 44, 1975, p. 509.
36. *Introduction to the IBM 3850 Mass Storage System (MSS)*, IBM Publication GA32-0028, 1974.
37. "Big Mass Storage Market Seen," *Electronic News*, June 23, 1975, p. 38.
38. *CONTROL DATA Mass Storage Facility*, Control Data Publication 201.197, April 1975.
39. *CalComp Newsletter*, California Computer Products publication, January/February 1975.
40. Damson, S., et al., "A Random Access Terabit Magnetic Memory," *Proc. Fall Joint Computer Conference*, 1968, p. 1381.
41. *TBM Mass Storage System*, Ampex Publication G414R, May 1975.
42. Gray, Edward E., "Laser Mass Memory System," *IEEE Trans. on Magnetics*, MAG-8, 3, p. 416.
43. Schwartz, David J. and Peter Gardiner, "MASSTAPE—A Systems Solution," *Proc. IEEE Computer Society Int'l. Conf. COMPCON74*, February 26-28, 1974, San Francisco, p. 109.
44. Schneidewind, Norman and Gordon Syms, "Mass Memory System Peripherals," *Proc. IEEE Computer Society Int'l. Conf. COMPCON74*, February 26-28, 1974, San Francisco, p. 87.
45. Speliotis, Dennis E., "Bridging the memory access gap," *AFIPS Conference Proceedings*, 44, 1975, p. 501.
46. Amelio, Gilbert F., "Charge-coupled devices for memory applications," *AFIPS Conference Proceedings*, 44, 1975, p. 515.

47. Ypma, John E., "Bubble Domain Memory Systems," *AFIPS Conference Proceedings*, 44, 1975, p. 523.
48. Hughes, W. C., et al., "BEAMOS—A New Electronic Digital Memory," *AFIPS Conference Proceedings*, 44, 1975, p. 541.
49. Greenblott, Bernard J. and Mu Y. Hsiao, "Where Is Technology Taking Us in Data Processing Systems," *AFIPS Conference Proceedings*, 44, 1975, p. 623.
50. *Systems Network Architecture,* IBM Publication GA27-3102, 1975.
51. *DECNET*, Digital Equipment Corporation Publication, 1975.
52. *Network Communications Systems for CDC Computer Systems and Data Networks*, Control Data Publication, 1975.
53. "Packet Switching Goes Commercial," *Computerworld*, June 11, 1975.

An evaluation of the East German RYAD 1040 system

by ROBERT A. KOENIG
Control Data Corporation
Minneapolis, Minnesota

ABSTRACT

In the early 1970's, several East European countries and the USSR developed a compatible computer family called the Unified System, or RYAD. In 1975, Control Data purchased the ES-1040, a member of the RYAD family for testing and evaluation. Performance and compatibility tests were made and the technology was assessed. Tests determined that the ES-1040 is compatible with IBM 360 instruction set. Benchmark programs show the ES-1040 to be twice as powerful as the IBM 370/145 in scientific/engineering applications, and at least as powerful in BDP work. The processor, using TTL IC's lags the U.S. by three years, the core memory shows a lag of about eight years, while peripheral equipment is eight to ten years behind the U.S.

INTRODUCTION

My purpose is to present an evaluation of the East European RYAD 1040 Computer System, and to share with you some of the significant results. Before we go into the details, some background is necessary to set the stage for further remarks.

In 1969, the five year plan for the CMEA (Council for Mutual Economic Assistance) countries specified the development of a computer family called the Unified System of Electronic Computing Techniques. This family is commonly referred to as the RYAD which is the Russian word for series. The CMEA countries consist of Bulgaria, Czechoslovakia, the German Democratic Republic, Hungary, Poland and the USSR. The development and production of the various models were delegated to the different countries, as was the peripheral equipment. In some cases, variants of a single model are produced in more than one country.

The family consists of six processors, numbered from 1010 to 1060, and a complement of peripheral equipment. The family uses an instruction set compatible with that of the IBM 360, and has common peripheral interfaces at the I/O channels. Because of the strong resemblance to the IBM 360/370, much of the evaluation is couched in terms of comparison to the IBM products.

In early 1975, Control Data purchased the largest available member of the RYAD family, an ES-1040 system made by the Robotron organization of the GDR. The system was received and installed at our Plymouth, Minnesota facility in July, and later moved to Washington, D.C. It is now installed in a CDC facility in Vienna.

THE ES-1040 SYSTEM

The system we received consisted of the hardware shown in Figure 1, together with operating software. The computer hardware consists of the central processor, an operator console, a main memory of 256 kilobytes of core storage, a byte multiplexer channel, and one selector channel. The peripheral equipment consisted of a card reader, card punch, line printer, two 7.25 MB disk drives, and two 79 IPS tape drives. I will discuss the hardware characteristics somewhat later.

We augmented the system by attaching some CDC peripheral subsystems normally offered in the plug-compatible market. The augmented system is shown in Figure 2. The specific items we attached, and used

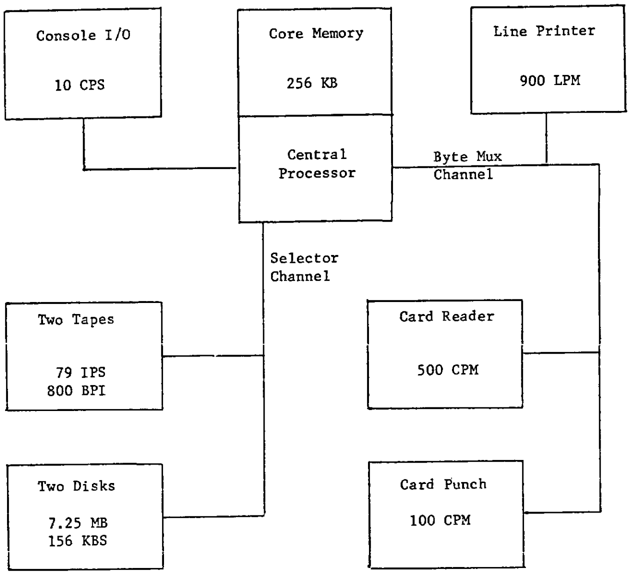


Figure 1

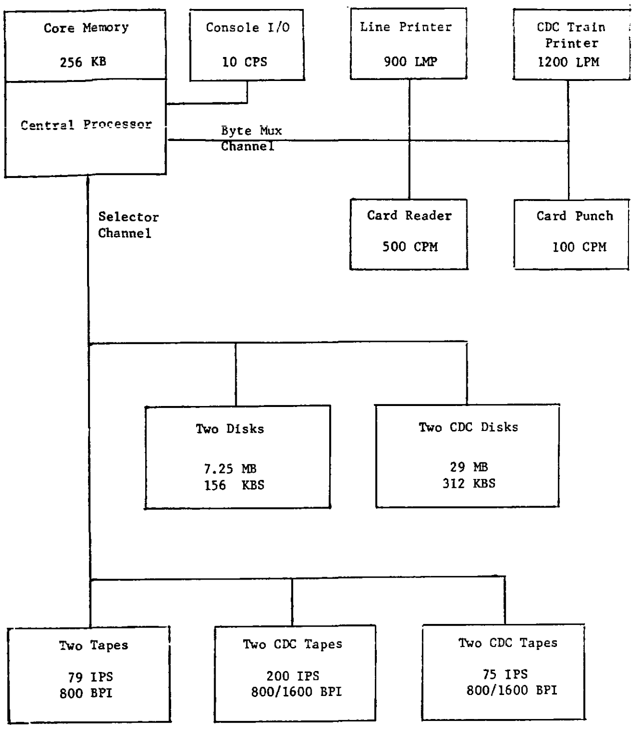


Figure 2

routinely with the ES-1040 consisted of a 1200 LPM train printer, two 29 MB disk drives, and two magnetic tape subsystems. One of these tape subsystems had two 200 IPS dual density tape units, while the other had two 75 IPS dual density drives. All these subsystems operated successfully after appropriate modifications were made.

COMPATIBILITY TESTING

As I noted earlier, the RYAD family uses an IBM 360 instruction set. To test this compatibility we ran a set of jobs from the Service Bureau Company. In this test, SBC personnel brought their operating system, recorded on a 2316 disk pack, customer programs and customer data. The SBC software, normally run on an IBM 360, was loaded into the ES-1040 via an IPL, and the customer programs were successfully executed. In our programming efforts, we routinely used IBM manuals, since the Robotron manuals were written in German, and English language translations were generally not available.

The peripheral equipment used compatible international standards. We interchanged magnetic tapes between the ES-1040 and IBM systems with no difficulty. We also were able to use a 1316 disk pack formatted for an IBM 2311 drive on the system, although we did not test interchangeability since we couldn't locate a 2311. The card equipment uses standard 80 column cards.

DESCRIPTION OF THE ES-1040

In general terms, the ES-1040 processor can be classified as a medium to large machine, falling between the IBM 360/50 and 360/65 in terms of compute power. It is an integrated circuit machine constructed from TTL devices produced in East Germany. These devices are mounted on 72-pack multilayer boards which average four to five layers. The backpanel is wirewrapped, although fully automatic techniques are not used in production. The general workmanship in the processor was very good, and the reliability extremely high. We experienced one hour of CPU downtime during five months of operation, and had no CPU card failures. It is the routine practice to turn off power at the end of each day, turning it on again the following day.

The operation of the processor is based on the use of microprogram control. The microprograms are stored in a ROM core memory containing 3K words of 130 bits each. This memory has an access time of 100 nanoseconds and a full cycle of 450 nanoseconds, the latter constituting a major timing cycle.

Insofar as registers are concerned, the ES-1040 looks very much like an IBM 360. For example, it has 16 general purpose and four floating point registers. Because of its compatibility with the 360, an instruction may consist of two, four, or six bytes. The repertoire contains 143 commands, of which 87 are basic, eight deal with decimal arithmetic, 44 are for floating point operations, two are for storage keys, and the other two are used for direct control.

There are some important differences between the ES-1040 and IBM 360 models as far as the internal design is concerned. As I will bring up later, the 1040 performance is hampered by a relatively slow memory. However, this is masked to a large extent by an instruction lookahead feature. This allows memory accessing, address modification, and instruction execution to be overlapped. When considering the basic TTL circuit speeds and the major timing cycle of 450 nanoseconds, the execution times of complex operations, such as divide, indicate that some rather sophisticated algorithms have been employed.

ES-1040 MEMORY

The ES-1040 uses a main memory constructed from 21 mil cores. It has an access time of 450 nanoseconds, and a cycle time of 1200 nanoseconds, although the effective systems-level time is three CPU cycles, or 1350 nanoseconds. In our machine, the memory is composed of two 128K independent modules. In a maximum system of one megabyte, four modules are used. The access path to memory is eight bytes, with a theoretical bandwidth of 142 megabits per second possible when three or more modules are used. The memory uses a single parity bit for each 8-bit byte.

I/O CHANNELS

As I mentioned earlier, the ES-1040 we received had one byte mux channel for attachment of low-speed devices and one selector channel. The I/O structure can be expanded to include one byte mux and up to six selector channels. The first selector channel has a speed of 1300 kilobytes per second. Channels 2 and 3 comprise Group 2 and have an aggregate rate of 1100 KBS, assignable to a single channel if only one is used, or split evenly between the two when both are installed. Similarly, channels 4, 5, and 6 have a total rate of 900 KBS, which can be used on one channel, or divided into 300 KBS increments, if more than one is used. These rates apply when all channels are active.

In general, the channel rates far exceed the capabilities of the peripheral equipment used with the system.

PERFORMANCE EVALUATION

I would now like to discuss some of the evaluation results. We concentrated most of our efforts in evaluating the performance of the processor, since I/O-bound jobs would be highly affected by the restrictions of a single selector channel and by the peripheral configuration. One of the first things we did was a Gibson mix analysis. As you are probably aware, this analysis evaluates CPU performance primarily in a scientific and engineering environment. The results are displayed in Figure 3 using 64 bits as the floating point mode. As can be seen, the ES-1040 is about twice as fast as the IBM 370/145, two-thirds the speed of the IBM 370/155, and about half the speed of a CDC CYBER 73.

We also wrote some FORTRAN programs designed to test the speed of the 1040 versus the IBM 370/145. The pure floating point arithmetic tests showed that the ES-1040 is three to four times faster than the 370/145. On the other hand, a FORTRAN program stressing memory speed, and using only integer arithmetic showed that the 370/145 ran that program at twice the speed of the 1040. This result confirmed what we expected from comparing specified memory speeds. We also ran a comprehensive FORTRAN test program submitted by a customer to our benchmark center. Neither the 370/145 nor the 1040 could successfully execute this program using single precision REAL variables. When converted to double precision, the 370/145 ran the program in 41 minutes, as compared to 20.3 minutes on the ES-1040, a ratio closely approximating the 2:1 obtained in the Gibson Mix.

MODEL	*VALUE*
IBM 360/50	6.50
IBM 370/145	4.95
ES 1040	2.40
IBM 370/155	1.77
CDC CYBER 73	1.23
CDC CYBER 173	0.78

Figure 3—Gibson Mix Values

We were not nearly as successful in testing the 1040 in BDP applications for several reasons. First, we did not have a COBOL compiler in the Robotron software product set, and secondly, we had only a single channel to service both tape and disk. We did write some assembly language programs using the variable length instructions. The results of these indicate that the ES-1040 will be at least as fast as the IBM 370/145 in BDP applications, provided it were similarly configured.

I should note that our evaluation would have been easier if we could have had a manual fully defining instruction timing. As far as we know, no such manual exists either in English or German.

SOFTWARE AND DOCUMENTATION

At the time we procured the ES-1040, we also purchased a set of operating software known as DOS/ES. This bears striking resemblance to IBM DOS. The product set under DOS/ES included a FORTRAN IV compiler, an assembler, RPG, and PL/1. Notably lacking was a COBOL compiler. It is interesting to note that all compiler diagnostics were printed in English, although certain systems messages and diagnostics were in German.

In addition to DOS/ES, a more advanced system, called OS/ES, is also offered. From the information offered, it appears to be very similar to IBM OS, offering MFT and MVT options. To our knowledge, no ES-1040 systems are using OS/ES, probably because of the lack of adequate mass storage.

The documentation was very mixed in quality. The programmer manuals available in English were inadequate to the point that we used IBM manuals in that area. The German language manuals were better, but still less than what one receives from a domestic manufacturer. The hardware manuals, primarily for CE use, were very thorough and complete, but exclusively in German. This was of little concern to us, since we had Robotron field engineers on-site to supply maintenance.

COMPARISONS WITH WESTERN TECHNOLOGY

There is a great deal of interest in comparing the State-of-the-art of Eastern Europe technology with that of our own. Based on our examination of the ES-1040 equipment, no single statement can be applied across the board. Quite clearly, the CPU is the most advanced element of the system. As I mentioned it is constructed from TTL devices with an U.S.-type numbering scheme. The devices used in the ES-1040 are relatively simple members of the family. By knowing when the equivalent devices were available in the U.S., we estimate that an American company could have completed a 1040 equivalent in 1968 or 1969. Since the first 1040 appeared in late 1972, we conclude that a time lag of three to four years exists in processor area.

The core memory attached to the 1040 is not a good match considering the power of the CPU. The technology is about eight years behind ours. The power consumption is about twice that of comparable memories used on domestic machines. The deficiencies of memory performance were apparently recognized early in the design cycle, and are effectively masked by the use of the instruction lookahead and memory interleaving features.

The peripheral area in general lags the U.S. by eight to ten years. The significance of the lag on general systems performance varies, of course, with the device. The one of most importance is that of disk drives. The two disks and the packs we received were made in Bulgaria, presently the only source of drives and packs in the CMEA countries. Although Bulgaria is also the primary disk controller supplier, the disk controller with the ES-1040 was produced in the GDR. The disks have 2311 characteristics. Its capacity is 7.25 MB, the transfer rate is 156 KBS, the rotational speed is 2400 RPM, and the average access time is about 65 milliseconds, even though a voice coil actuator is used.

Disk drives of the 2311 class went into production in the U.S. over 10 years ago, and since that time we have seen three new generations evolve. As far as the 1040 is concerned, the disks are clearly inadequate, not only for the obvious reason of capacity, but also the other performance parameters.

The ES-1040 tape subsystem, built in the GDR consisted of a controller and two 79 IPS drives, using 800 BPI nine-track format. The tape drives used dual capstans with pinch rollers, an old design approach. The tape loading process was very tedious and required considerable manual dexterity. The U.S. was producing comparable tape units over 10 years ago, and since then we have increased the bit density by a factor of eight and the tape speed by a factor of 2.5.

The line printer and associated controller were built in the GDR. It uses drum printer technology, operates at 900 LPM, and has 156 columns. It contains a unique feature; in effect a split platen, which allows two separate forms to be controlled independently. Thus, the printer can appear as two logical units to a programmer. This feature is supported by the Robotron software. Despite the technological drawbacks of the printer, it is adequate for most uses.

The card I/O equipment was made in the USSR, apparently having been in production for 10 to 15 years. The card reader reads 500 CPM, photoelectrically, while the card punch operates at 100 CPM, punching a row at a time. With the de-emphasis on card I/O, these units are adequate for most cases.

In examining the equipment, one quickly becomes impressed by its ruggedness, and high labor and material content. Workmanship ranges from good to excellent. Despite some old engineering designs, the equipment was very reliable. Preventive maintenance consisted of a daily PM of one to two hours and one weekly PM of four hours. Included in the daily PM was checking the adjustment of the positioners on the disk drive. All PM and EM work was performed by Robotron field engineers assigned to our system in accordance with their normal practices.

SUMMARY

I would like to conclude my remarks by making the following points:

- First, the ES-1040 demonstrated instruction set compatibility with the IBM 360 line.
- Second, the ES-1040 processor is about twice as powerful as the IBM 370/145 in scientific and engineering applications and at least equivalent to the 370/145 in BDP applications.
- Third, general processor technology seems to lag the U.S. by three or four years.
- Fourth, the core memory is about eight years behind.
- Finally, there is a significant lag in peripheral capability, which we estimate at eight to ten years.

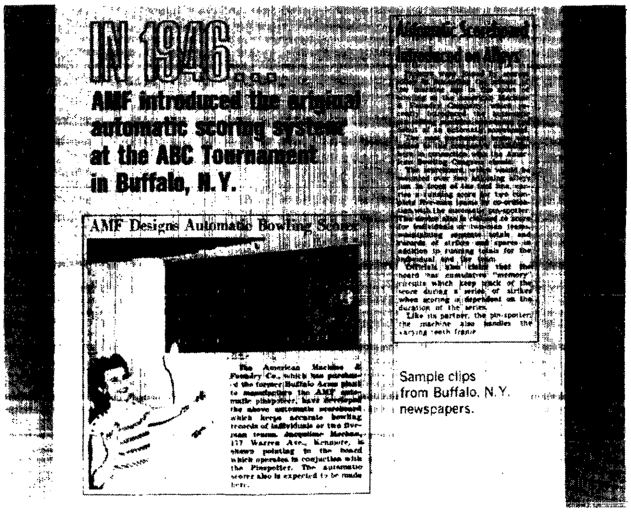
MagicScore bowling scorer—A microprocessor application for fun and profit

by REG A. KAENEL
AMF Incorporated
Stamford, Connecticut

ABSTRACT

Microprocessors have made the implementation of automatic bowling scorers practical. A specific scorer design will be described and illustrated, development considerations associated with this scorer will be discussed, and the future of microprocessor applications will be projected.

Automatic scoring has finally come of age. It has been around for a long time. At stake is a market represented by some 150,000 bowling lanes in the United States alone. AMF Incorporated demonstrated the first working model of a mechanical score board in 1946 when it presented the first automatic pinspotter at the ABC tournament in Buffalo NY (Figure 1). Automatic score keeping was way ahead of its time and far from being economically viable in those days.



IN 1946...
AMF introduced the original automatic scoring system at the ABC Tournament in Buffalo, N.Y.

AMF Designs Automatic Bowling Scorer

The American Machine & Foundry Co., which has purchased the former Buffalo Arms plant to manufacture the AMF automatic pinspotter, have developed the above automatic scoreboard which keeps accurate bowling records of individuals or two five-man teams. Jacqueline Mechan, 117 Warren Ave., Kenmore, is shown pointing to the board which operates in conjunction with the Pinspotter. The automatic scorer also is expected to be made here.

Automatic Scoreboard Introduced on Alleys

[illegible]

The scoreboard, which would be mounted over two adjoining alleys just in front of the foul line, carries a running score for two complete five-man teams in co-ordination with the automatic pin-spotter. The device also is claimed to score for individuals or two-man teams, maintaining separate totals and records of strikes and spares in addition to running totals for the individual and the team.

Officials also claim that the board has cumulative "memory" circuits which keep track of the score during a series of strikes when scoring is dependent on the duration of the series.

Like its partner, the pin-spotter, the machine also handles the varying tenth frame

Sample clips from Buffalo, N.Y. newspapers.

Figure 1—Sample clip from Buffalo, NY newspapers reporting on AMF's automatic score board (1946)

Advances in semiconductor technology made automatic score keeping increasingly more practical ever since then. Accordingly, three different scorers were introduced two decades later with solid-state electronic control. The Automatic Scorer by Brunswick Corporation and the ScoRite scorer by Itek subsidiary, Doban Laboratories, both used a printing projector, by which scores were mechanically impact-printed on transparent pre-printed forms which were projected onto a screen through an optical system. An experimental model of the MagicScore bowling scorer system by AMF used CRT displays. Detection of standing pins was accomplished electromechanically with both the AMF and Brunswick systems using existing pin light switch-mechanisms of the automatic pinspotters. The Itek system used an array of photodetectors and directional light sources to perform this pindetection function. Each Brunswick console served two pairs of lanes, each AMF and Itek console one pair of lanes, but the AMF consoles were controlled from a central unit (Figure 2) for up to 30 consoles shown in Figure 3. Brunswick was the only company which immediately began production of their scoring system. The recent emergence of the microprocessor provided the technological advance necessary for making viable the implementation of a sophisticated automatic self-contained scorer system (Figure 4).

Additional electronic scorers have been introduced since the demonstration of those first ones. They included the semiautomatic Scortronic System by Sharp, the Automatic Scorer by Ikegami Tsushinki, the Meltas system by Mitsubishi Electric, the Automatic Bowling System by Kinetic Systems and the automatic Rapid-Score System by RCA. During this period AMF began marketing a semiautomatic scorer (i.e., EasyScore) on an interim basis to bridge the gap until the viable fully automatic MagicScore bowling Scorer system became ready for production. Pinfall data is entered by hand through keyboards with the semiautomatic systems in contrast to the automatic systems where data can be keyed in by hand if need be but which data normally is entered electrically from suitable pin-detectors. All recently introduced systems have self-contained player consoles, except for the RCA system which uses a central processor and a central printer; all these systems use CRT display devices for clarity and simplicity of operation. The AMF and RCA systems provide for a manager console by which various administrative control functions can be performed over player consoles, such as game monitoring, establishing a practice mode of operation, and displaying the statei of these consoles.

It is generally acknowledged today that bowlers prefer automatic to hand scoring primarily because it gives more time to socialize and to relax. Hand scoring is viewed by many bowlers as a disagreeable task which requires concentration and makes the scorekeeper subject to criticism for making an error. Of hand scoring's two major elements, doing the arithmetic is more objectionable than recording the pinfall, particularly among women. The reasons why arithmetic is disliked are the concentration required, the fact that some bowlers feel inadequate regarding arithmetic, and the fear of making a mistake. This last reason also applies to recording pinfall. Hand scoring is considered a nuisance and a distraction from bowling involvement. It is not that scoring is unimportant, it is simply that bowlers feel involved in the game without manually keeping score.

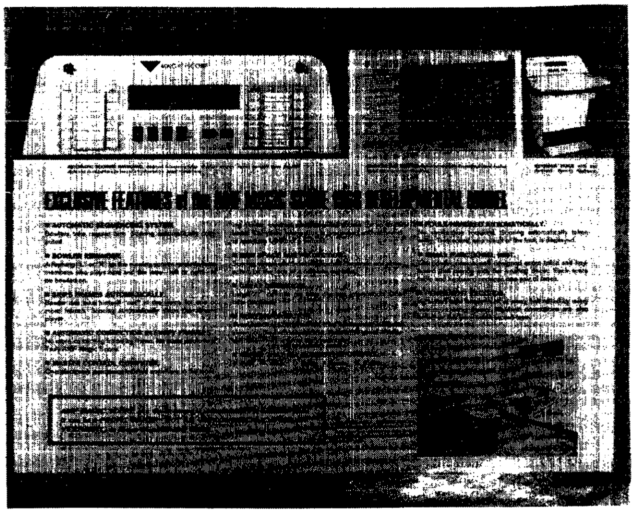


Figure 3—High-lights of AMF's experimental MagicScore bowling scorer contained in a 1968 advertisement flyer

Figure 2—Central electronic 30-console control of AMF's experimental MagicScore bowling scorer system

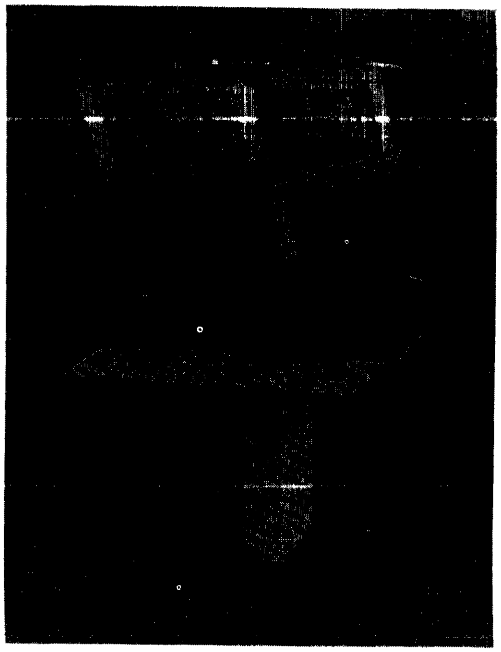


Figure 4—Today's self-contained MagicScore bowling scorer console made possible with microprocessors

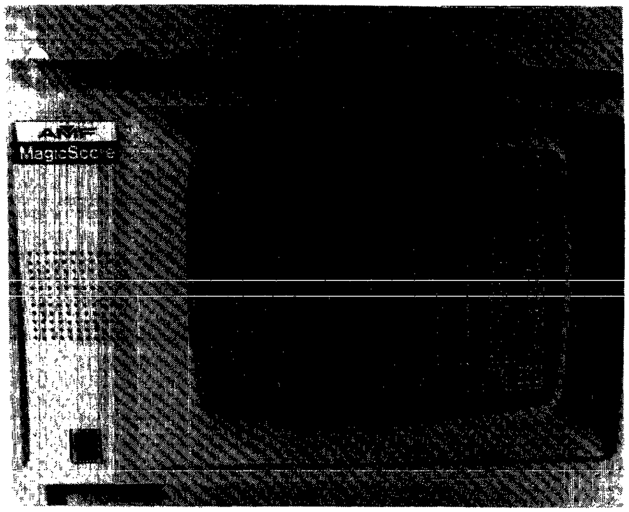


Figure 5—Close-up view of the CRT display, showing display format and character font

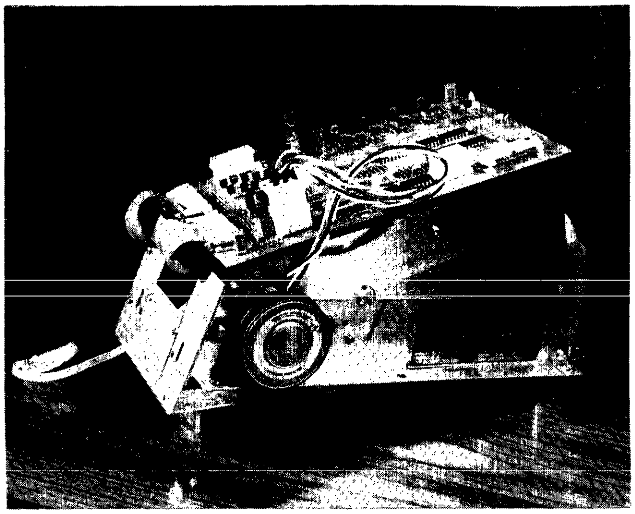
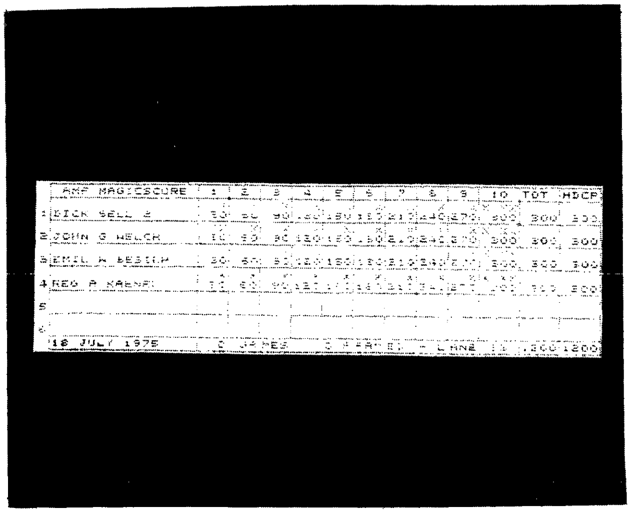


Figure 7a—Photograph of thermal printer, revealing thermal print-head subassembly, paper-advance stepping motor

OUTLINE OF FUNCTIONAL SUBSYSTEMS

The basic format of the bowling score sheet, as approved by the American Bowling Congress, best depicts the progress of a game of bowling. Accordingly, it was decided that the bowling scorer display subsystem emulate its key features, (Figure 5) as well as the printer subsystem (Figure 6) with which permanent records of games are produced. A cathode-ray tube display medium was selected for its outstanding brightness and clarity. To maximize size of the pinfall-data characters (Figure 5), each bowler is designated by three letters and the frame-by-frame subtotals were omitted on the display. A novel thermal printer was found to be most cost-effective, reliable, and maintenance-free since it has only one moving part (i.e., the paper-feed stepping motor) and only requires replenishment of thermally-active blank paper (Figure 7); a total print-out, which includes running subtotals by frames (Figure 6), is produced after the game ends. Display characters are made up from dots of a 5×7 matrix with a non-interlaced horizontal raster-scan where the same pattern of dots appears on adjacent pairs of scan lines; in contrast, printed characters use 5×5 dot matrix. Extra print-head dot-elements are provided for producing the grid-lines of the print-outs.



	AMF MAGICSCORE	1	2	3	4	5	6	7	8	9	10	TOT	HDCP
1	DICK BELL 2	30	60	90	120	150	180	210	240	270	300	300	300
2	JOHN G WELCH	30	60	90	120	150	180	210	240	270	300	300	300
3	EMIL W BESICH	30	60	90	120	150	180	210	240	270	300	300	300
4	REG A KAENEL	[illegible]	60	90	120	[illegible]	180	210	240	[illegible]	300	[illegible]	300
5													
6													
	18 JULY 1975	[illegible] GAMES		[illegible]				LANE [illegible]				1200	1200

Figure 6—Copy of Sample-Printout

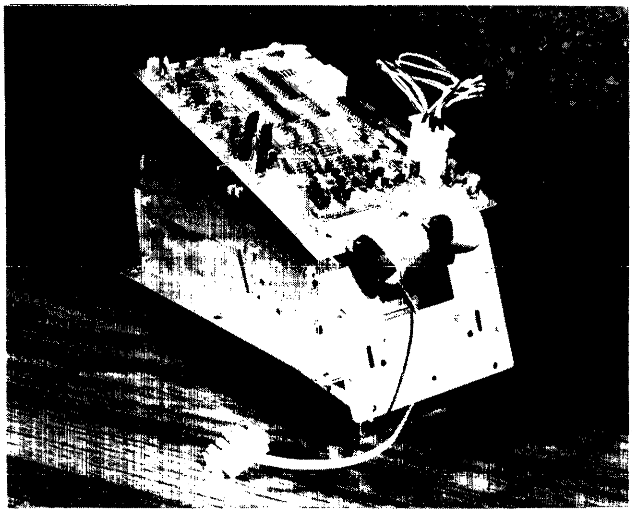


Figure 7b

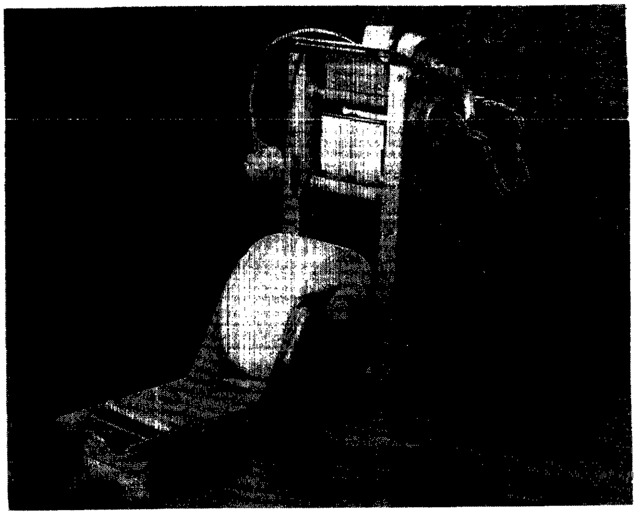


Figure 7c

The following data must be defined at the outset to initialize a game of bowling: names of the players in bowling sequence, handicap of each bowler, and the designation of pacers whose score is not added to the team totals or blind bowlers who will not physically bowl but whose game average score is entered during initiation. The scorer derives representative pinfall data from the average score entered, and displays it when it is the blind player's turn to bowl as if this player actually bowled. In addition, the system must be placed in either the open or league mode of sequencing; in the open playing mode the bowlers assigned to a lane bowl in a round-robin fashion, but in the league mode the bowlers of a pair of lanes are sequenced to bowl alternately on both lanes. Initialization of the scorer is performed through a center keyboard shared by the players of the pair of lanes served by the particular scorer (Figure 8).

Ruggedness, flat sealed construction, and tactile feel were guiding requirements for the development of this and the other scorer keyboards.

During the initialization operation, the operator first selects the sequencing mode (i.e. Open or League) which becomes indicated on the console by LED indicators. A particular player-row is then chosen by designating left or right side, if the proper side has not already been selected, followed by selecting the proper row. This selection will be visually acknowledged by a cursor that appears over the number of the display row. Next, the field is selected in which data will be entered from the keyboard. Depressing the Player Name key produces a cursor on the bottom row below the 6th player row where the next name character entered will be displayed. Up to 16 characters are accepted and displayed on the bottom row; but only the first and the next two characters that are preceded by a space are displayed in the name field of the player row selected. In contrast, the full 16 character name appears in this name field of the print-out. All cursors are removed when a new side selection is made; no cursors are reproduced on the print-out since they are used only to cue the bowlers during data entry. Sequence mode selection becomes inhibited when the first pinfall data is entered; at that time, a player cannot be unbinded or un-pacered.

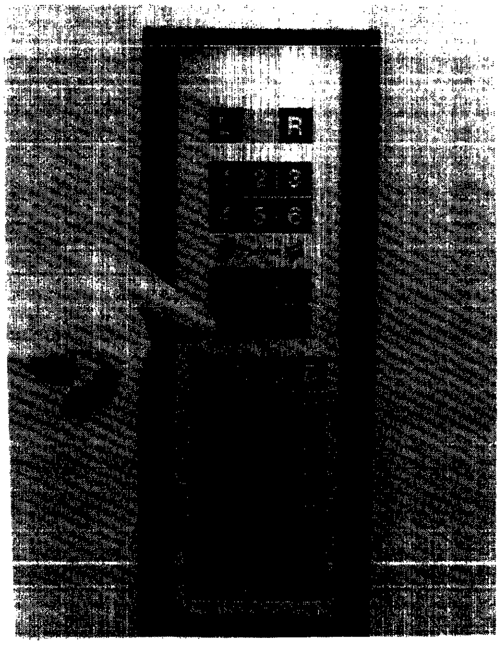


Figure 8—Close-up view of the center section of the keyboard

Being fully automatic, the bowling scorer begins cueing players to bowl once the system has been initialized with the data of the first players. The players who are to bowl are called by their full names which are displayed on the bottom display line on the lane-side they are to bowl on. In addition, arrows appear in the fields where the next pinfall data will be entered, pointing to the lanes from which the data will be taken.

Two extra keyboards, dedicated each to one lane-side, are available to provide for out-of-order player-sequencing and error correction of pinfall-data (see Figure 9). Forcing a player in out-of-sequence is accomplished by depressing the desired row-number key on the home-side of the team the player belongs to.

The MagicScore bowling scorer can be operated as an option in the semiautomatic mode by setting a concealed switch to that mode. In this mode pinfall data is entered by way of the score-key array located on the

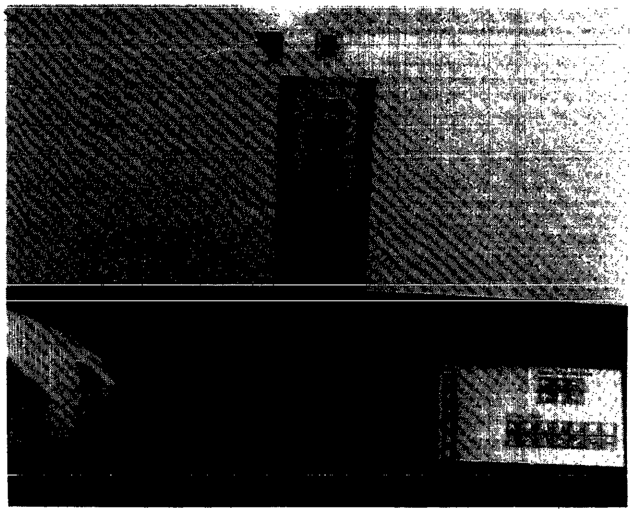


Figure 9—Close-up view of the Console Keyboards, showing center keyboard and dedicated additional keyboards

lane-side where the pinfall data originated. Other than that, the keyboards serve the functions outlined above.

OUTLINE OF THE ELECTRONIC LOGIC

The electronic scorer logic is contained on two printed-circuit boards which are mounted behind the key-board subassemblies on the front panel hinged for ease of access (Figure 10). The printer drive electronics is contained on a printed-circuit board mounted on top of the printer subassembly itself to make it even more of a readily replaceable functional unit. The electronic logic is partitioned onto a microprocessor board (left side board as seen in Figure 11) and a display processor board (right side board of Figure 11). The overall logic schematic is shown in Figure 12.

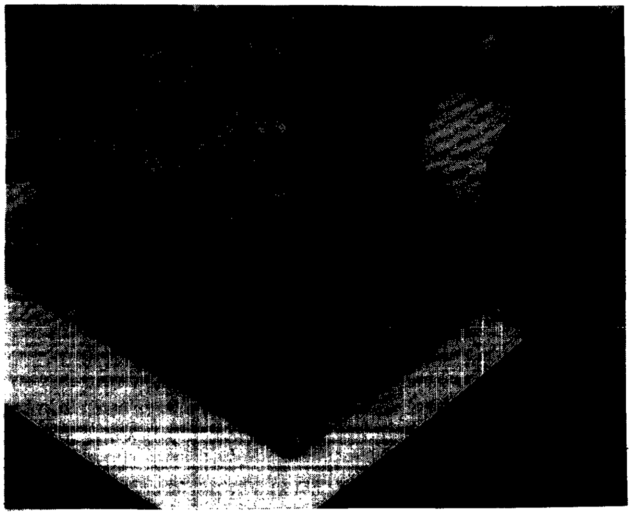


Figure 10a—Console open, revealing printing subassembly, power-supply, MPU electronics board Display Processor electronics board

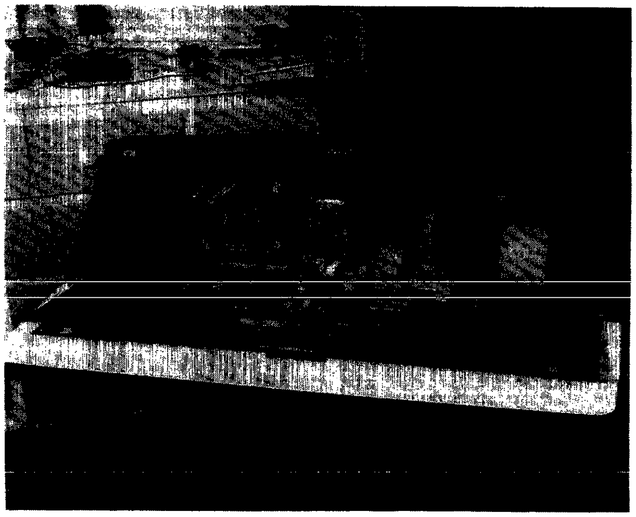


Figure 10b

The microprocessor board contains an M6800 type microprocessor with three Peripheral Interface Adapters of the M6800 family of components, one Asynchronous Communications Interface Adapter which connects to a common serial communications bus, and five 2000×8 bits read-only memories which store the scorer control software. It also includes a six-position DIP switch array by which each scorer can be assigned

Figure 11—Close-up view of the MPU electronics board (left) and Display Processor board (right)

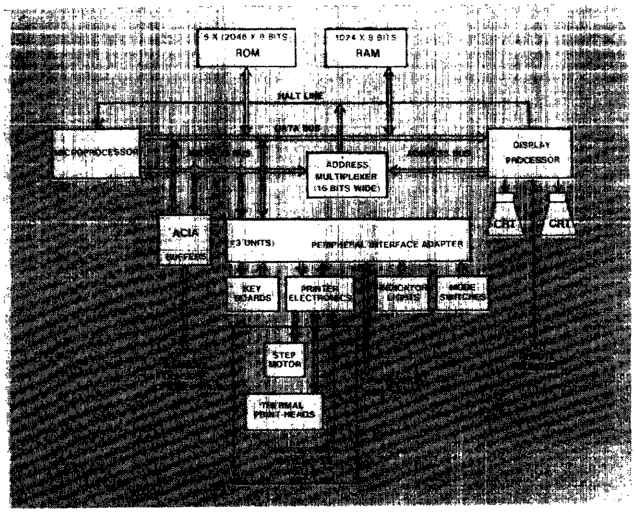


Figure 12—Simplified block diagram of the MagicScore bowling scorer logic

a unique address which it uses to recognize messages on the communications bus. The display processor board contains the random-access memory of the scorer in addition to the circuits that produce suitable video signals for the display monitors (Figure 13). A rechargeable standby battery is provided for these random-access memories to preserve pinfall data for many hours during power failures.

The scorer software, whose flow-chart is shown in greatly simplified form in Figure 14, provides for processing score data from the keyboard (Figure 9), the pinsensor, and the manager console. The pinsensor receive routine creates a compatible pinfall data word from the incoming width modulated data (i.e., a binary zero is represented by a pulse of duration 10×8 ms, the left-lane delimiter pulse by a pulse of duration 16×8 ms, and the right-lane delimiter by a pulse of duration 22×8 ms; binary ones and zeros are serially accumulated in a shiftregister as they arrive and this word is forwarded to the pinfall-data processing routine when a delimiter is detected), executes the manager-console subrouting (i.e., read, interpret, and execute data received by the asynchronous communications interface adapter), increment timers (e.g., time-out for clear key-activation and for indicator flash control), and activate indicators. An executive program sequentially performs such functions as producing signals for the printer, calculating score subtotals, computing the next bowler to play and cueing him, etc.

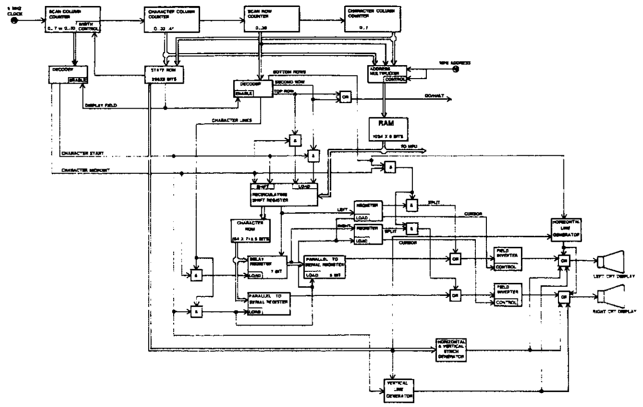


Figure 13—Block diagram of the Display Processor

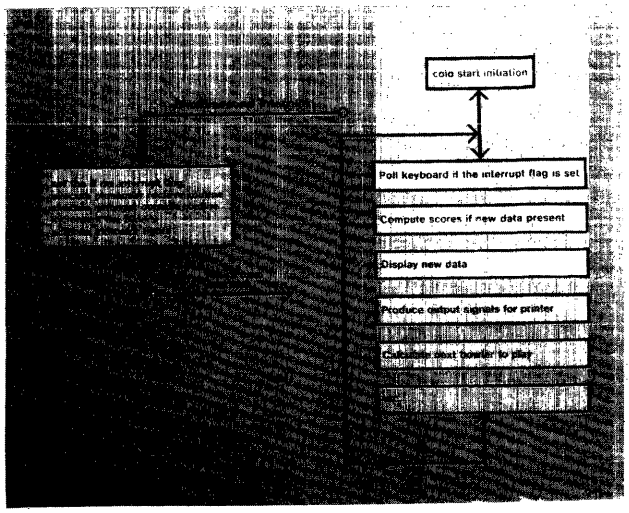

colo start initiation
Poll keyboard if the interrupt flag is set
Compute scores if new data present
Display new data
Produce output signals for printer
Calculate next bowler to play

Figure 14—Flow chart of the software

During program execution the microprocessor deposits display and other data in the random-access memory of the Display Processor board and retrieves such data from this memory. Concurrently, the display processor electronics fetches data from this memory and processes it for display (Figure 13).

Each row of display characters is made up from 28 scan lines. A complete line of display characters from both display sides is transferred from the random-access memory into a recirculating shiftregister during the first two scan-lines of each character row, and recirculates there for the full duration of this row. The characters of the left screen are located at even addresses of the random access memory, those of the right screen at odd addresses; points to the left and right side display characters are similarly interspersed in that shift-register. A character column counter designates the column number of the character that is being scanned across by the cathode-ray tube beam at every instant. It is used to read the left-side data from the random access memory during the top scan line of each character row and the right-side data during the second scan line.

As the character count is advanced, the left-side character is converted into a suitable pattern of seven dots by means of a character read-only memory. This pattern is stored in the left-side buffer shiftregister. The same process is immediately repeated for the right-side characters. These dots are then serialized and outputted to the respective display monitors.

A state read-only memory is used to define the width of each character (i.e. seven counts when part of a string of characters and ten counts when followed by a vertical line), the presence of a horizontal gridline, the presence of a horizontal synchronization signal, etc.

Means are also provided for selectively directing either the left-side video signals or the right-side signals to a video bus of the manager console. When both sides are selected, this is decoded to blank out the display monitors. This switching function is provided by the manager console which has access to all of addressable space (including RAM and peripheral interface adapters) through the microprocessor by way of the asynchronous communications interface adapter.

The component efficiency afforded by a microprocessor is evident (Figure 11). Not so is the extent by which the use of a microprocessor facilitated the developmental task: operational aspects of the control software were defined as the hardware was being developed, and operational refinements were incorporated while the system was being tested. The flexibility of a microprocessor based design made incorporation of ancillary functions both compelling and practical. The manager console feature is a case in point. It was decided to provide flags in random-access memory by which a manager can remotely control the bowling scorer consoles via a serial communications bus. The functions made available through these consoles include blanking out scorers, monitoring lane activity through respective display, disabling the clear function at a console so as to preserve the count of games and frames bowled, and projecting messages onto an individual display screen.

Operational details are described in the User Instructions attached to the MagicScore Bowling Scorer which will be demonstrated during the presentation of the paper at the conference.

QLISP—a language for the interactive development of complex systems

by EARL D. SACERDOTI, RICHARD E. FIKES, RENE REBOH,
DANIEL SAGALOWICZ, RICHARD J. WALDINGER
and B. MICHAEL WILBER
Stanford Research Institute
Menlo Park, California

ABSTRACT

This paper presents a functional overview of the features and capabilities of QLISP, one of the newest of the current generation of very high level languages developed for use in Artificial Intelligence (AI) research.

QLISP is both a programming language and an interactive programming environment. It embeds an extended version of QA4, an earlier AI language, in INTERLISP, a widely available version of LISP with a variety of sophisticated programming aids.

The language features provided by QLISP include a variety of useful data types, an associative data base for the storage and retrieval of expressions, the ability to associate property lists with arbitrary expressions, a powerful pattern matcher based on a unification algorithm, pattern-directed function invocation, "teams" of pattern invoked functions, a sophisticated mechanism for breaking a data base into contexts, generators for associative data retrieval, and easy extensibility.

System features available in QLISP include a very smooth interaction with the underlying INTERLISP language, a facility for aggregating multiple pattern matches, and features for interactive control of programs.

A number of applications to which QLISP has been put are briefly discussed, and some directions for future development are presented.

INTRODUCTION

An important byproduct of research in artificial intelligence (AI) has been the development of programming languages that permit instructions to be given to a computer at a very high level. A second important byproduct has been the development of highly sophisticated, supportive interactive programming environments. Tools of this kind are very important for developing AI programs, which tend to be large, complex, and subject to frequent alteration. As the needs of the computing community grow, and the ability of hardware to perform more calculations in a given time improves, we believe the programming tools that have been a necessity to AI will become important tools of general applicability.

This paper will present a functional overview of the capabilities and features of QLISP, one of the newest of the current generation of very high level AI languages that includes MICROPLANNER,[1] SAIL,[2] CONNIVER,[3] POPLER,[4] and others. As such, it will serve both to introduce the language to the computing community, and to provide a brief overview of the features available in the new generation of AI languages. A more extensive treatment of QLISP is available elsewhere.[5]

QLISP is both a programming language and an interactive programming environment. It grew out of the QA4 language[6] that was developed at the Stanford Research Institute from 1969 through 1972. Many of the basic concepts of the language are derived from the QA4 work. QLISP embeds an extended version of QA4 in INTERLISP,[7] a widely available version of LISP with a variety of sophisticated programming aids. In addition, it provides many new features not present in other languages.

In the following section, we will describe the language features of QLISP, with special emphasis on those not available in other languages. (Bobrow and Raphael[8] give a comparative description of a number of these languages.) Then we shall describe the programming environment provided by QLISP and the underlying INTERLISP. Finally, we shall give some examples of the ways in which the language has been used to create complex software systems.

LANGUAGE FEATURES

This section will discuss the more notable features of the QLISP language. Most of these are derived from features present in QA4. Some are derived from other languages. Most have been extended for greater ease of use, compatibility with the underlying INTERLISP language, or increased generality.

Data types

QLISP provides a very rich set of data types and facilities for manipulating them. In addition to the range of types provided by INTERLISP (including numbers, arrays, strings, list and binary tree structures), QLISP provides data of type TUPLE, VECTOR, BAG, and CLASS.

A tuple is similar to a LISP list, but can be accessed via associative retrieval as described below. A vector is like a tuple, but is treated somewhat differently when evaluated.

A bag is a multi-set, an unordered collection of elements with possible duplicates. For example, (BAG A A B C) is equivalent to (BAG A C B A) but is different from (BAG A B C). Bags are particularly useful for describing the argument lists of associative commutative relations. For example, if we defined the relation PLUS to take a bag as its argument, then the expressions (PLUS A A B C) and (PLUS A C B A) (which would both be stored internally as (PLUS (BAG A A B C))) would be equivalent by definition.

A class is an unordered collection of elements, without duplication. For example, (CLASS A A B C) is equivalent to (CLASS C B A).

Associative data base

Expressions composed of any of the data types mentioned above may be placed in a data base. The data base is designed for *associative retrieval*, the fetching of data by content rather than by name or address. A request for an item of data may specify values for any of its constituent elements, leaving the rest to be matched by the values in the retrieved item. The data base is maintained in the form of a discrimination net, a tree-like structure in which the nodes represent tests to apply to an expression, and the branches represent the values returned by the tests. In general, these tests are set up to find the first difference, scanning left to right, between two expressions.

Canonical representation of expressions

By storing all data in a common discrimination net, QLISP can represent equivalent expressions uniquely. In the QLISP net, only one instance of an expression may occur. Before an expression is entered into the net, it is transformed into a canonical form. A new datum will not be created if the expression already occurs in the net. Thus, continuing our example about the PLUS relation, (PLUS A A B C) and (PLUS A C B A) are not only equivalent; they are exactly the same pointer into the data base.

Property lists

Arbitrary expressions are represented uniquely in QLISP, just as atoms are represented uniquely in LISP. Therefore it is possible to assign properties to QLISP expressions in the same way as LISP atoms. For instance, we may execute the command

```
(QPUT (PLUS A B (MINUS A))
   SIMPLIFIESTO B),
```

which will put the value B under the indicator SIMPLIFIESTO in the property list of the expression (PLUS A B (MINUS A)). If this expression, or any equivalent expression (such as (PLUS B (MINUS A) A)), is ever encountered again, we can look on its property list and find a simplification for it.

One particular indicator on the property lists of expressions is used to represent truth value. When this indicator, MODELVALUE, has a value T, the system interprets that expression to be "true." Similarly, a value of NIL represents a "false" expression. Special statements exist for manipulating this particular property. For example, the statement

```
(ASSERT (AT SRI MENLO-PARK))
```

would simply place the attribute-value pair (MODELVALUE T) on the property list of the tuple (AT SRI MENLO-PARK)*. The semantics of the statement is that SRI is in Menlo Park. Similarly, the statement

```
(IS (AT ←THING MENLO-PARK))
```

would cause a search of the data base for something that was known (i.e., was in the data base with MODELVALUE equal to T) to be in Menlo Park.

The unification pattern matcher

An important activity in AI programs is the construction, modification, and analysis of complex symbolic expressions. The most powerful tool for this is a *pattern matcher*, an algorithm that allows one expression to be used as a template to break up another expression into components. QLISP extends this facility by providing a *unification* pattern matcher in which each of two expressions may act as templates for the other.

Some examples at this point are appropriate. The

* This paper will avoid almost all need for the reader to cope with QLISP-specific syntax. It suffices to say that in QLISP statements, the elements of expressions are presumed to be constants unless identified as a variable by the prefix ← or $. The ← prefix indicates that the variable is to be assigned a new value; the $ prefix indicates the previous value of the variable.

QLISP statement MATCHQQ invokes the pattern matcher directly. The statement

```
(MATCHQQ (←X←Y) (A B))
```

will match X to A and Y to B. The statement

```
(MATCHQQ (←X←X) (A B))
```

will fail, since X cannot be bound simultaneously to A and B. The statement

```
(MATCHQQ (A←X) (←Y B))
```

will match X to B and Y to A. The statement

```
(MATCHQQ (A (B←X) ←Y)
    (←X←Z (A(B C))))
```

will match X to A, Y to (A (B C)), and Z to (B A).

The QLISP pattern matcher is based on an extended unification algorithm that can deal with the variety of data types available in the language. The matcher is not complete for complex expressions containing bags and classes. However, it is adequate for the kinds of expressions that are almost always used. Pattern matching is used in QLISP for several central purposes. It is used to bind variables and decompose expressions, as we have mentioned. It is used to control associative retrieval. It is also used to invoke functions for specified purposes, as we will now show.

Pattern-directed function invocation

Many of the AI languages provide a feature, first proposed by Hewitt,[9] whereby functions can be invoked not only by naming them, but also by checking to see if they are appropriate for a given argument. This check is performed by matching a pattern associated with each function with the given argument. For example, we might write some functions for an algebraic simplifier that looked like this*:

```
PLUSSINGLE:  (QLAMBDA (PLUS←X) $X)
PLUSZERO:    (QLAMBDA (PLUS 0 ←←X)
             ('(PLUS $$X)))
PLUSMINUS:   (QLAMBDA (PLUS←X
             (MINUS←X) ←←Y)
             ('(PLUS $$Y)))
```

The PLUSSINGLE function says: given an argument of the form PLUS followed by any single element, return that single element. The PLUSZERO function says: given an argument of the form PLUS followed by any number of elements, one of which is 0, return the form PLUS followed by all the other elements of the argument.

At the user's option, if a function's pattern can match an argument in more than one way, all possible matches may be attempted in turn. When one match leads to a failure, an alternative match is attempted. The function itself will not fail until all possible matches have been tried. For example, the following program will find two friends of JOE who are father and son:

```
QLAMBDA (FRIENDS JOE (CLASS←F←S
         ←←REST))
        (IS (FATHER $S $F))
        BACKTRACK)
```

The program will cycle through all pairs of elements from the class of JOE's friends and see if one is the father of the other.

Teams of functions

Functions to be invoked by pattern are typically defined for application toward a specified purpose. Some functions are to be used for consequent reasoning: when a particular consequence or goal (characterized by the function's pattern) is desired, invoke this function to achieve it. Some functions are to be used for antecedent reasoning: when a particular antecedent condition (e.g., an assertion in the data base) (characterized by the function's pattern) occurs, invoke this function to cause further effects on the data base.

Typically, all the consequent functions are tried when a goal is to be achieved, and all the antecedent functions are tried when the data base is updated. Only the ones that have a pattern that matches the goal or assertion will actually be invoked, but a great deal of overhead must be expended to attempt to match the patterns of all functions.

This practice is inefficient since many functions may already be known to be inappropriate, and yet their patterns will all be checked. QLISP provides a feature whereby, with each of many kinds of statements that can invoke functions by pattern, a so-called *team* of functions can be specified from which applicable ones may be drawn. So, in our simplification example, we could cause one simplification to occur with a statement that calls for consequent reasoning:

```
(CASES (PLUS A 4 (MINUS A))
    APPLY (PLUSSINGLE PLUSZERO PLUS-
    MINUS . . . )).
```

The list after the keyword APPLY is the team of functions associated with the particular CASES statement. The system will attempt to match the patterns of only these functions with the particular PLUS expression.

Similarly, a team of functions may be specified with any ASSERT, DENY, DELETE, or QPUT statement to perform antecedent-type activities. For example, in

* The doubled prefixes (e.g., $$) indicate that the variable refers to a fragment of the expression containing it rather than a single element. The quote mark (') indicates that the following expression is to be instantiated (following the semantics of QLISP) rather than evaluated (following the semantics of LISP).

a computer system modelling the operation of a library, a team of functions might be associated with assertions that modelled a book being checked out. These functions might assert that the book was due in three weeks from the current date, update a count of books in circulation, or even cause the original assertion to fail, and provide appropriate other action to be modelled if the person checking out the book had overdue books outstanding. This activity could be initiated by a QLISP statement of the form:

```
(ASSERT (CHECKEDOUT (The Odyssey)
        James.Joyce (4 JAN 1918))
        APPLY $LIBRARYFNS
```

where $LIBRARYFNS was bound to the list of relevant antecedent functions.

Contexts

The previous discussion has presumed that all expressions were stored in a single, monolithic data base. In fact, the data base is factored into different segments, called *contexts*. Contexts may be thought of as corresponding to the block structure of ALGOL-like languages. Whenever a QLAMBDA function or a user-defined block is entered, the current context is set to be a descendent of the previous context. Variable bindings and assignment of properties (including, in particular, truth values) to expressions that are local to a context are perceivable only from that context or some descendent. Thus, contexts may be regarded as particular viewpoints on the data base.

In addition to a default structuring of contexts based on the structure of the flow of program control, QLISP provides facilities for manipulating contexts explicitly. For example, to prove a proposition of the form:

(P or Q) implies R,

one could set up two parallel contexts with P true in one and Q true in the other, and try to prove R in both contexts, as suggested in Figure 1.

Contexts are actually constructed from more elementary entities, which we shall call *contags*, for want of a better term. Contags, which are similar to the "situation tags" of PLASMA,[10] correspond to particular points in time in the evaluation of a program (typically when QLAMBDAs or blocks are entered). A context is an ordered list of contags, typically corresponding to the stack of active function and block invocations. For the user with sophisticated needs for data base manipulation, we have provided a set of QLISP statements that permit a user to construct his own contexts, or viewpoints on the data base, from the underlying contags. These statements allow the creation of a context that is a descendent of a number of independent contexts, a context that is the subset of a given context not retrievable from a second context, and a context that revises a given context to appear as if it were descendent of another arbitrary context.

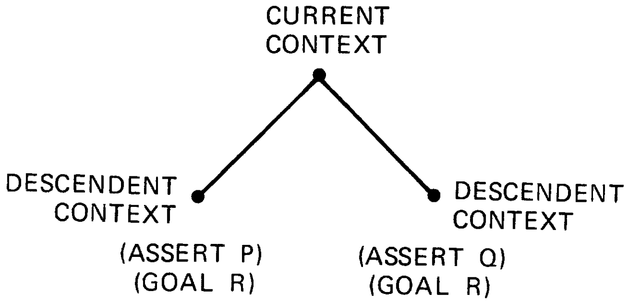


Figure 1—Using contexts to prove a disjunction

Generators

The data retrieval statements of QLISP are designed to find a single instance of a given pattern. To cause the pattern matcher to continue its search and obtain other such instances, a user's program must return to the query statement via the backtracking mechanism (i.e., by failing).

To allow a more natural and inexpensive method of retrieving multiple instances of a pattern, we have extended the CONNIVER[3] approach of using *generators*. For example, the IS statement that was introduced earlier specifies the retrieval of one instance of a given pattern. There is a generator version of the IS statement called GEN:IS that finds multiple true instances of a given pattern. Each time a statement such as GEN:IS is called, it produces a number of instances matching the pattern. These expressions are put on a "possibilities list" along with a "tag" that indicates how the generator can be restarted when more instances are requested, and this possibilities list is returned by the generator as its value.

If the function TRY:NEXT is called with a possibilities list as an argument, it will remove the first instance from the list and return it as its value. If the list contains no more instances, the tag is used to restart the generator. Since calls to TRY:NEXT can be made from anywhere in a program, this form of generator separates the retrieval of data elements from the processing that is done on them in a way that is not possible in a strict backtracking regimen.

The generator retrieval statements are implemented using INTERLISP FUNARGs. A FUNARG is a data object that conceptually represents a copy of a function together with that copy's private data environment.

Extensibility

QLISP statements that are not part of the underlying INTERLISP language are processed by the

INTERLISP error handling mechanism, as will be explained below. User-oriented tools for accessing the LISP translation mechanism are provided so that new QLISP-like statements can be defined easily. Once the statements have been defined, they are treated by the interpreter and compiler exactly like other QLISP statements. Typically, the extension facility has been used to provide alternative control structures for invoking the standard QLISP statements, or to provide special syntax for user-defined QLAMBDA functions.

SYSTEM FEATURES

QLISP is more than just a programming language; it is an interactive programming environment for the development of very complex collections of software. In this section we shall discuss the major features of this environment that are unique to QLISP.

Integration with INTERLISP

The major advantage of QLISP as a programming environment, as compared with other new AI languages, is its ease of use. It is easier to edit functions, create symbolic files, trace execution paths, break into computation paths, and debug programs in QLISP. This is primarily due to the choice of INTERLISP as the host language for QLISP, and the care that has been taken in the implementation of QLISP to preserve the many supportive features of INTERLISP.

QLISP is implemented via the error handling mechanism of INTERLISP. A valid LISP expression will never be seen by the QLISP processor. Thus, programs or portions of programs that use only LISP constructs will run as fast in QLISP as in INTERLISP.

When the INTERLISP interpreter encounters an ill-formed LISP expression, it calls an error routine that in turn invokes an error analyzer. If the expression is recognized as a valid QLISP form, it is translated to an equivalent LISP form that is returned to the interpreter for evaluation. The translation is stored with the original expression so that the translation need be done only once.

A similar mechanism causes QLISP code to be translated into equivalent LISP code when it occurs within a function being compiled. Since the translation occurs at compilation time, the QLISP interpreter need never be invoked at all when running compiled QLISP code.

Aggregation of pattern matches

The "apply team" mechanism provides a good means of reducing the number of unneeded pattern matches during pattern-directed function invocation. However, there may still be a lot of wasted effort as the function invocation mechanism attempts to match each pattern in turn to the argument. For example, the simplification functions described in the preceding section all begin with PLUS. They might even be segregated into a specific team of functions to simplify expressions beginning with PLUS. And yet every pattern will be matched against the argument, and the matcher will succeed at least as far as matching up the PLUS's.

An option is available to allow the patterns of QLAMBDAs to be aggregated together in a tree structure. For example, the tree for the simplification functions listed earlier appears as in Figure 2. A single operation against the tree can determine the set of all the QLAMBDA functions that are good candidates to successfully match a given argument. (The tests that are applied are cruder than those applied by the pattern matcher itself, so that the set of functions may contain some whose patterns won't actually match when the matcher is invoked.) This set can then be intersected with the particular "apply team" to determine which functions to invoke.

The tree structure that is used is actually the discrimination net that is used as the associative data base. For "apply teams" of more than fifteen functions or so, this feature provides significant efficiencies.

Interactive program control

Since the QLISP backtracking mechanism is implemented using INTERLISP's error facility, there are a number of ways in which the standard INTERLISP interactive facilities won't work properly. For ex-

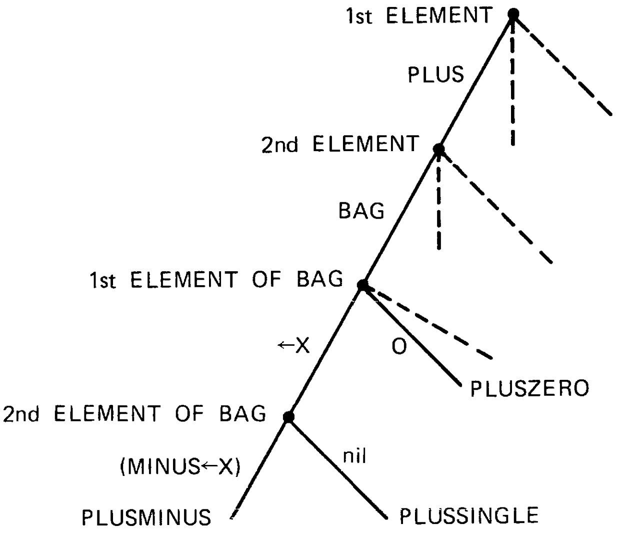


Figure 2—Pattern selection tree for simplification functions

ample, the INTERLISP function tracing facility is implemented as "break the computation, then print, then continue." But INTERLISP errors, which are generated by QLISP to cause backtracking, are trapped at a break. The solution we adopted to this particular quandary was to implement a QTRACE facility that did not generate a break when it printed information about a function invocation. Similar care was taken with breaks in computation, the package for manipulating symbolic files, and many other system components to allow a QLISP user to believe that the total system was behaving exactly as the underlying INTERLISP would.

APPLICATIONS

To provide some perspective on the utility of QLISP, we will briefly describe some of the applications to which it has been put. The common characteristics of these applications are that the programs and the concepts underlying them could not be specified without a sustained cycle of programming the best current ideas about what the program should be doing, observing the program's behavior, and then modifying or extending the ideas.

Program verification

The first major QLISP program was the program verifier of Waldinger and Levitt.[11] The verifier was originally written in QA4. The program includes over 100 functions each encapsulating a specialized piece of knowledge about the semantics of the language of the programs being verified. The QLISP version ran about 30 times faster than the original QA4 program.

Automatic programming

Subsequent work by Waldinger has used QLISP for the generation of simple programs from output specifications. This work makes strong use of the unification feature of the pattern matcher to combine the knowledge that is distributed in various QLAMBDA functions. For example, one function may say, in effect, to produce a list with some X as its CAR, perform (CONS X something), where the something is unspecified. Similarly, another function may say, to produce a list with some X as its CDR, perform (CONS something X), where the something is again unspecified. So if the system were given the goal of producing a list with A as its CAR and B as its CDR, the first function would return (CONS A something), the second would return (CONS something B), and by unifying these results the system can produce the correct code (CONS A B).

General problem solving

A system developed by Sacerdoti[12] generates complex plans, monitors their execution, and recovers from unexpected events that cause the execution to deviate from the expected course of action. This is a large system (about 60 pages of code) and almost all of it is in the underlying INTERLISP language. However, the pattern matching and context mechanisms of QLISP are central to its operation, and the ease with which the representation of knowledge could be changed was important in the system's development.

The semantics of the actions that the system plans for are written in a language extension of QLISP that has QLISP's semantics but is evaluated very differently. Strong use of the pattern matcher and of the extension feature allowed the action language to be readily changed as the scope of the program increased.

Deductive retrieval

A deductive retrieval package was written by Fikes[13] to allow arbitrary deductions to be fired off by a QLISP-like query. In addition to simply causing associative retrieval from the data base, Fikes' queries can fire off arbitrary programs to deduce the answer to the query from other available information. These query statements, implemented as a language extension of QLISP, make strong use of the generator facility. Capabilities for modelling state changes, also part of this package, make strong use of the ability to associate an arbitrary property list with an expression.

Computer aided design

The first general-purpose program for computer aided design that uses AI techniques in a substantial way has recently been completed. It works by generating a model of the object to be designed in stages of increasing detail. As each stage is generated, appropriate user-supplied design constraints are applied. The system employs sophisticated backtracking techniques to minimize the search for an object that satisfies all the constraints. The program in its current form[14] is written completely in INTERLISP. The development of the program was carried out in QLISP, and the code was gradually cut over to pure INTERLISP as design ideas gelled and execution speed became important. The pure INTERLISP version runs about ten times faster than the original QLISP version. The development of the system was greatly facilitated by the early use of QLISP and the resulting ability to easily change internal representations and control strategies.

Econometric modelling

A system has been developed that integrates a quantitative computer model with an overlay of heuristic judgmental rules.[15] The heuristic overlay is intended to facilitate interactive use of the econometric model by making it easy to alter parameters and adjust boundary conditions. The underlying quantitative model was implemented in a mixture of INTERLISP and FORTRAN. The heuristic model was implemented in QLISP as an ASSERT team, a set of functions applied after an assertion has been made in the model. The user interface was implemented in QLISP because of the ease of interaction it provides.

CURRENT STATUS

QLISP has been in active use at SRI for nearly two years. The version at SRI is implemented in INTERLISP on a PDP-10 computer using the TENEX operating system. It is available for use over the ARPAnet by other users on the network.

A version of QLISP is also available for INTERLISP-370, a version of INTERLISP that runs on IBM 360 and 370 series computers.

QLISP is not intended to be a performance language. The programming tools that it provides are of general purpose. Thus a program written in QLISP will run slowly compared to a version of the program written in a language that provides a more restricted set of data types or less flexible control structures. But it has been our experience that, when the programs to be written are large, complex, and subject to frequent alteration as development proceeds, then the inefficiency in the program's execution time is more than compensated for by efficiencies in the programmer's development time.

FURTHER WORK

While QLISP is a useful tool for many purposes, further work will be required to augment the power of the language to reflect the growing needs of AI programming. The current version provides an associative data base that must be entirely contained within the program's core image. Systems that operate on substantial knowledge bases are a focus of current research interest in AI, and the amount of data that these systems will use will require that at least part of the associative data base be resident on secondary storage. This will require a new data storage and retrieval mechanism, since those of existing AI languages, including QLISP, tend to distribute data randomly throughout the store. The distribution of data needs to be at least partially based on semantic criteria, instead of being totally on a syntactic basis as is done now.

Another inadequacy of QLISP and other AI languages is that the pattern matcher returns too little information. Given two patterns to match, it replies either with an exact matching between the patterns or with a report of failure. It would often be extremely useful to have a measure of how "close" the match was to succeeding. Obviously, this would be an expensive feature, but this kind of "fuzzy" matching would provide a user with the powerful ability to begin to deal with expressions on a semantic basis.

A third area for further development is in the general category of multiprocessing. While many languages support the use of multiple interdependent processes, the level of command that they provide is typically quite low. Typically they are on the level of "start process," "suspend process," and "wait on semaphore." It would be very advantageous to have higher level commands available that would allow the language system itself to keep track of many processes at many levels of function calls. Such a mechanism could be easily tied to the existing "APPLY team" facility of QLISP.

CONCLUSIONS

We have given a brief overview of the capabilities and features of QLISP. While it is not practical for use as a production language, it is a time-saving tool for use in the construction of complex systems that are subject to significant change during the course of their development.

ACKNOWLEDGMENTS

The basic features of QLISP are derived from the QA4 language, developed at SRI by Jeff Rulifson, Richard Waldinger, and Jan Derksen. The initial implementation of QLISP was done by Rene Reboh and Earl Sacerdoti. Subsequent development was carried on by Daniel Sagalowicz and Mike Wilber. Rich Fikes developed the generator package, and was instrumental in straightening out the context mechanism. Mal Newey wrote the pattern matcher, based on a problem reduction algorithm of Richard Waldinger. Richard Waldinger has been a major force in setting the goals of the language development. Warren Teitelman has provided much help in generalizing the features of INTERLISP that permit the clean interface to QLISP.

REFERENCES

1. Sussman, G. J. and T. Winograd, *MICROPLANNER Reference Manual*, MIT Artificial Intelligence Laboratory, Memo No. 203, July 1970.
2. VanLehn, K. A., ed., *SAIL User Manual*, Stanford Artificial Intelligence Laboratory, Memo. AIM-204, July 1973.
3. McDermott, D. V. and G. J. Sussman, *The CONNIVER*

Reference Manual, MIT Artificial Intelligence Laboratory Memo No. 259, May 1972.

4. Davies, D. J. M., *POPLER 1.5 Reference Manual,* University of Edinburgh, TPU Report No. 1, May 1973.
5. Wilber, B. M., *The QLISP Reference Manual,* SRI AI Center Technical Note 118, March, 1976.
6. Rulifson, J. F., R. J. Waldinger and J. A. Derksen, *QA4: A Procedural Calculus for Intuitive Reasoning,* SRI AI Center Technical Note 73, November 1973.
7. Teitelman, W., *INTERLISP Reference Manual,* Xerox Palo Alto Research Center, October 1974.
8. Bobrow, D. G. and B. Raphael, "New Programming Languages for Artificial Intelligence," *Computing Surveys,* Vol. 6, No. 3, September 1974.
9. Hewitt, C., *Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot,* MIT AI Memo No. 251, April 1972.
10. Hewitt, C., "How to Use What You Know," *Proceedings of Fourth International Conference on Artificial Intelligence,* pp. 189-198, September 1975.
11. Waldinger, R. J. and K. N. Levitt, "Reasoning About Programs," *Artificial Intelligence,* Vol. 5, No. 4, pp. 235-316, 1974.
12. Sacerdoti, E. D., *A Structure for Plans and Behavior,* SRI AI Center Technical Note 109, August 1975.
13. Fikes, R. E., "Deductive Retrieval Mechanisms for State Description Models," *Proceedings of Fourth International Conference on Artificial Intelligence,* pp. 99-106, September 1975.
14. Latombe, J.-C., "Artificial Intelligence in Computer-Aided Design: The TROPIC System," Preprints of *IFIP Working Conference on CAD Systems,* Austin, Texas, February 1976, (Proceedings to be published by North-Holland under the title "CAD Systems").
15. Coles, L. S., "The Application of Artificial Intelligence to Heuristic Modelling," *Proceedings of Second USA-Japan Computer Conference,* pp. 200-207, August 1975.

User interface design issues for a large interactive system

by RICHARD WILLIAM WATSON
Stanford Research Institute
Menlo Park, California

ABSTRACT

User interface design issues are discussed for a large interactive system. The assumptions about the user environment are explicitly described. Issues discussed include command language syntax, command recognition and completion, subsystem organization, user extension capabilities, user options, and various forms of prompting, help and feedback. These issues are discussed within the context of an existing system, the NLS system.

INTRODUCTION

The large interactive system user interface issues discussed in this paper reflect experience at Stanford Research Institute (SRI) over the past twelve years in the evolution of the user interface to the NLS system. NLS is a prototype collection of tools in a growing workshop of tools and services to aid knowledge work.[1,4] NLS provides facilities to support activities such as document creation, study and publication, message handling, information filing and retrieval, and software engineering. We expect the number of tools and the vocabulary that controls the use of these workshops to grow. We further expect that the use of such workshops will spread throughout those occupations involved with information in various forms and that there will be infrequent and casual users of such systems, along with many people who will spend large fractions of their day using such workshops. One goal is to match the speed of system responsiveness to the natural speed and flow of man's thought processes. It is from these basic expectations that our user interface work has developed.

The sections below enumerate several assumptions and areas of concern around which the NLS user interface has developed to date. A key point to mention is that we do not consider the NLS user interface a static, finished product. It will change, based on analysis of usage experience, and the technology and media available.

The user interface has two sides: the input side by which the user inputs information, indicating by various conventions and controls what he wishes accomplished; and the output side by which the machine provides feedback and other assistance to the user in command specification, and provides various forms of information portrayal. Man has many motor and other capabilities that could be the basis for input and command specifications; similarly he has his full range of senses that could be targets for system output.

To date, computer information systems make use of only a few motor and sensory capabilities in their man-machine dialog. An important area of research involves exploring the advantages to be gained and the techniques to be used to extend this range. There is interesting research going on in areas of speech, eye movement, brain wave control, hand written script, and video graphics that will undoubtedly be integrated into the truly multimedia systems to be built in the near future.

We call the user's collection of input-output equipment, and arrangement of work tables and work space, the workstation. At the present time, input to interactive systems centers around various types of keyboard devices: standard typewriter-type, function button, keyset (chord), and graphical pointing devices (mouse, electronic pen-tablet, light pen, joystick). The dominant output means are teleprinters and displays of varying capabilities.

The present NLS user interface has been developed around this equipment, although many of the principles used in its design can be easily extended for use with other media.[3] The prime motivation for the use of the mouse for pointing and two keyboards (standard typewriter-like and keyset) as the input devices for the display version of NLS (DNLS), are described in References 2 and 3. NLS can also be used from typewriter terminals (TNLS). In this paper, we concentrate on describing some of the motivations behind the design of the NLS command language and the forms of information portrayed to assist the user in command specification. Forms of general NLS information portrayal are described in Reference 1.

HIGH LEVEL ASSUMPTIONS UNDERLYING THE DESIGN OF THE NLS USER INTERFACE

First we describe a few high-level assumptions about the system usage environment that affect the user interface design and then discuss some of the lower level issues and the specific techniques used to deal with them.

Coordinated set of user interface principles

There will be a common command interaction discipline, over the many application areas in the workshop, that shapes user interface features, such as the language, control conventions, methods for obtaining help, and computer-aided training.

This commonality has two main implications. One, it means that while each domain within the core workshop area or within a specialized application system may have a vocabulary unique to its area, this vocabulary will be used within language and control structures common throughout the workshop system. A user will learn to use additional functions by increasing vocabulary, not by having to learn separate "foreign" langauges. Two, when in trouble, he will invoke help or tutorial functions in a standard way.

Grades of user proficiency

A once-in-a-while user with a minimum of learning will want to be able to get at least a few straightforward things done. In fact, even an expert user in one domain will be a novice in others. Users will be clerical workers, information specialists, executives, engineers, and others. Attention to novice-oriented and to tutorial help features is required.

Users also want and deserve the reward of increased proficiency and capability from improvements in their skills, their knowledge, their conceptual orientation to the problem domain and to their workshop's system of tools, methods, conventions, etc. "Advanced vocabularies," short concise control notation and conventions in every special domain will be important and unavoidable.

A corollary feature is that workers in the rapidly evolving augmented workshops should be involved continuously with testing and training in order that their skills and knowledge may most effectively harness available tools and methodology.

Ease of communications between subsets and addition of workshop domains

One cannot predict which domains or application systems within the workshop will want to communicate in various sequences with which others, or what operations will be needed in the future. Thus, results must be easily communicated from one set of operations to another, and it should be easy to add or interface new domains to the workshop. A corollary is that the total workshop may contain a very large number of tools and services. Some users may have access to only a subset of its capabilities while others will have access to many or all capabilities.

As described below, we expect the workshop to be embedded in a computer network and thus communication between tools and between users must take place across both process and host boundaries according to well specified conventions and protocols.[5,6]

User programming capability or user interface extensibility

There will never be enough professional programmers and system developers to build or interface all the tools that users may need for their work. Therefore, it must be possible, with various levels of ease, for users to add or interface new tools, and extend the user language to meet their needs. They should be able to do this in either a variety of programming languages with which they may have training, or in the basic user-level language of the workshop itself.

Range of workstations and symbol representations

The range of workstations available to the user will increase in scope and capability. These workstations will support use of text with large, open-ended character sets, pictures, voice, mathematical notation, tables, numbers, and other forms of knowledge. Even portable hand-held consoles will be available. Indeed the multiplicity of possible terminals raises the question of whether a consistent set of control and portrayal conventions is possible.

As hardware decreases in cost, more and more capabilities will be placed in the workstation both in the form of user interface aids and facilities, and in the form of frequently used tools.

Distributed nature of the user interface processes

The collection of facilities to support interfaces with the system of tools can be conceived of as a single service as seen by the user. These facilities may all reside in a processor in the workstation or be distributed in two or more processors, depending on the level of their sophistication and state of the art with respect to cost, hardware capability, and so forth.

Tools embedded in a computer network

The computer-based tools of a knowledge workshop will be provided in the environment of a computer network, such as the ARPANET.[7] For instance, the core functions will consist of a network of cooperating processors performing special functions, such as editing, publishing, exchanging documents and messages, data management, and so forth. Less commonly used but important functions, such as a compiler, might exist on a remote machine. The total computer-assisted workshop will be based on many geographically separate systems.

Once there is a "digital-packet transportation system," it becomes possible for the individual user to reach out through his processor to other people and other services scattered throughout a "community." The "labor marketplace" where he transacts his knowledge work will be literally independent of geographical location.

Specialty application systems will exist in the way that specialty shops and services now do—and for the same reasons. When it is easy to transport the material and negotiate the service transactions, one group of people will find that specialization can improve their cost/effectiveness, and that there is a large enough market within reach to support them. And, in the network-coupled computer-resource marketplace, there will be a growth of specialty shops, such as application systems specially tailored for particular types of analyses, or for checking through text for spelling errors, or for doing the text-graphic document typography in a special area of technical portrayal, and so on. There will be brokers, wholesalers, middle men, and retailers.

The key point to emphasize is that even when hardware costs decrease to the point where a user can perform 90 percent of his work using tools and information that operate in the processor in his work station, he will want to have access to a computer network to:

(a) Communicate in various forms with others
(b) Access very large or special databases
(c) Access special tools that run elsewhere

Problem orientation of the command language and tolerance for ambiguity

The user has a task that he wishes performed by the system. Depending on the nature of the task and operations available to him on the system, he may be able to express what he wants accomplished in a single "statement" or command to the machine, or it may require a series of commands.

One of the goals of the designers of the command language and system is to understand the nature of the user's application domain so that the user can express his needs with constructs that are similar to his thought processes, natural problem solving vocabulary, and language forms. The machine will then break down the request into smaller steps as required for internal processing.

If there is ambiguity in the user's command, the machine should recognize it, if possible, and prompt appropriately for clarification. There is still much research and development required to fully meet this goal.

Many people hope to allow natural language to be used in making statements to the machine. This capability will require models of the user and task domains for understanding.

Even when systems are able to interpret commands given in natural language, the precision and usage efficiency of appropriate artificial languages will make the latter's continued use preferable, especially for skilled users.

Given the above general considerations as background, we can move on to examine features of the NLS user interface in more detail.

SOME COMMAND LANGUAGE CONSIDERATIONS

A command language must allow unambiguous specification of what the user wishes accomplished. The operation to be performed, and the entities or information items (arguments) to be acted upon, or used to determine what is to be acted upon, must be specified. These can be specified in a variety of ways: by typing them in full or in some form of abbreviation, by pointing at them on a screen, by pronominal reference, by implication from context, or by use of default values automatically assumed by the system where appropriate. The order of their specification, the syntax or grammar of the language, can have various forms. For example, operational command-words can be specified, followed by the arguments, or vice versa. Arguments can be in fixed positions or explicitly named and occur in any order. Some arguments or command-words can be optional and require special characters to indicate their presence. Arguments or command-words can have defaulted values under certain conditions. Pronominal references can be allowed to refer to previous occurrences. Arguments may be given types by the system and language designer for more extensive error checking and feedback.

Arguments and keywords can be specified by complete or partial typein (there are a variety of forms of command recognition that are discussed later) or designated by pointing to representations on a display or by use of specially coded function keys. Or, the machine may ask questions and the user just fill in the blanks.

Depending on the characteristics of the computer and communications system, it may or may not be possible to provide command word or keyword completion, prompting or other feedback, argument checking, default value fill in, and so forth, during the command specifications.

For example, in line-at-a-time, half-duplex systems, the user usually must complete the entire specification of the command before transmission to the system, while in character-at-a-time, full-duplex systems, the system can react to each character received and provide more extensive aids to the user during command specification.

The above discussion outlines just a few of the many choices available to the language designer. As the purpose of this paper is not to be a complete tutorial on all possible choices available and their advantages and disadvantages, the following discussion gives only the main NLS command language features and the motivation for their adoption.

THE NLS COMMAND LANGUAGE

The NLS command language generally has the following form, where angle brackets group meta symbols:

⟨operation specification⟩ ⟨operand specification⟩ ⟨command completion⟩

The fields in a command are of a fixed order, although some commands have optional fields that can be specifically requested. Other fields can have a system-supplied default value. Because NLS operates from a character-at-a-time, full-duplex system, several levels of help are available, as described later, for giving cues and prompting, explicitly listing options or syntax, and giving full documentation on what the system expects next during command specification. It was not felt that much would be gained for novice users by allowing fields to be specified in any order by using explicit field names. Novice users do not need to be aware of optional fields.

As much as possible NLS makes the operational specification of the form verb-noun followed by arguments and possibly other keywords. We have also tried to maximize the fullness of the verb-noun matrix.

This approach seemed to be natural, and follows normal English imperative forms to aid learning. The choice of verb-noun form seemed to fall out naturally when considering such important areas as editing. A given verb or operation, such as DELETE, can naturally be applied to many entities, such as STATEMENT (a paragraph, title, equation), CHARACTER, NUMBER, TEXT, FILE etc. Learning is easier if the user can form a model of how the system works that can be consistently applied. In this case, a user can learn n verbs and m nouns and understand that generally, if it is meaningful, they can be used in pairs. Having learned n+m vocabulary terms, he can apply them in the form of n×m commands. For example one can command DELETE STATEMENT, DELETE NUMBER, DELETE FILE etc.

We have tried to pick command keywords that have normal usage related to the operation described. A synonym capability would be easy to implement.

Four forms of command keyword recognition are provided to enable the user to choose the one most appropriate to his terminal type, system response, previous system experience, and present NLS experience level. We have worked to pick an operational vocabulary for the present system that guarantees keywords to be unique in a maximum of three characters:

(1) A single-character mode allowing high-speed single-character recognition of the most commonly used command keywords; less commonly used command keywords require an escape character followed by enough characters for unique recognition: With large and expanding command sets one cannot choose keywords with mnemonic value and guarantee uniqueness in the first character. This mode is generally preferred by experienced users because of the conciseness and speed with which frequently used operations can be expressed. We find that experienced users are very concerned that commands be formed with the minimum number of input operations, and that commands have the richness needed to specify adjective or adverb type operations as needed. There is thus some conflict in certain commands between these goals for the experienced user and the need for command simplicity for the novice.

(2) A demand mode requiring a special character to initiate recognition: This has proved to be popular for new users of typewriter terminals, particularly those with experience using the TENEX operating system, under which NLS currently runs.[13]

(3) An anticipatory mode requiring the user to type just enough characters for the command to be uniquely specified; the system then automatically fills in the remainder of the command word.

(4) A fixed mode that guarantees recognition on entry of three characters.

Given the implementation approach outlined later, it is quite easy to add other recognition modes, such as allowing the user to choose keywords from a menu displayed on the screen. However, experiments have shown that the time it takes to point to an item on the screen is equivalent to several keystrokes and thus would be disadvantageous to skilled users, although possibly of value to novices.[2,3]

Modes 3 and 4 have turned out not to be heavily used.

Operand argument specification is contained in a number of fields that are variable with the type of command. All commands of a similar type have the order of the operands as consistent and as natural (relative to normal English usage) as possible. Infrequently used operand fields are optional and novice users need not be aware of their existence.

Related to argument specification is the problem of choosing argument delimiters. There is a need for the following delimiting functions.

(1) Delimiting command words

(2) Delimiting arguments
(3) Delimiting optional arguments or command word fields
(4) Delimiting commands
(5) Selecting arguments from a display screen, and confirming the selections

One could choose separate characters (codes) to represent each of these functions. To do so seemed to us to add an unnecessary complication for the user. Therefore, except for using a special character to indicate an optional argument, or command word, a single code is used for the other functions in NLS. We call this code "Command Accept" (CA) even though it is used for other purposes as well. The system allows the user to define which keyboard character is to serve this function if he finds the system default to be inconvenient. One of the buttons on the mouse also serves this function.

Arguments can be typed in, defaulted where appropriate, or specified by pointing to appropriate entities on the display screen.

There are three flavors of command completion.

(1) Command Accept: Completion of the command indicating execute the command and return to the base state to await input of the next command. The default indication for this form is one of the buttons on the mouse in DNLS, which is translated into a control character. Command completion is defaulted to be CR in TNLS. The use of CR in TNLS is quite natural and generally does not conflict with textual input as most text in NLS is typed in without explicit CRs and is appropriately formatted by the system for various output devices. If the TNLS user wishes to input an explicit CR in his text file, he must precede it with an escape character. If he has need to enter many CRs in his text string, he can redefine the completion character, Command Accept, to be some other character.

(2) Repeat: Completion of the command and return to an appropriate intermediate command state for quick repetition of the command. Repetition mode continues until explicitly commanded to escape out of it. This mode is very useful when a delete or other operation is repeated several times.

(3) Insert: Completion of the command and entry to insert-statement mode for addition of new paragraphs or other text statements. This mode is like command repeat above except that it always takes you to the insert command. It is used frequently when one adds, replaces, or moves text, and then wants to follow it with new statements. It speeds text input when inserting sequences of paragraphs.

The system is to be used from a variety of terminal types, including both typewriter-type terminals and displays. The two-dimensional displays are to be the preferred workstation types whenever a design decision must be made between language forms possibly favoring one type or the other.

We decided to make the command language syntax for the TNLS version and the DNLS version as close as possible, except where the difference between the one-dimensional and two-dimensional media would clearly prohibit this or would seriously limit one or the other version. This decision allows people working in environments consisting of both typewriter and display terminals to move back and forth with ease.

The system has been organized into clearly defined subsystems with uniform rules for their entry and exit. Any subsystem can be entered from any other, either to "execute" a single command with automatic return or to perform a chain of commands. The user can return either to a specifically named subsystem in the path of subsystems traversed, or enter a new subsystem. The issue of how to group commands into subsystems has to do with training and patterns of use rather than system constraints. It relates to learnability and, to some extent ease of command specification using single characters as switching subsystems switches vocabularies, and to "knowing where you are" in a command or operational space.

One could construct a system where all commands were in a single subsystem. Study of the command set of a large system particularly conceived of as a set of tools shows that operations tend to group together in such a way that to perform a given task, such as sending a message or calculating a budget, generally require several related suboperations. Certain operations, such as moving in information space or seeking help, tend to be used as suboperations of many or all tasks. This latter observation has led to "universal" commands available from within any subsystems. One can also imagine certain commands to be needed frequently in just two or more subsystems and thus implemented in each subsystem having the need. There are now no instances of this case in NLS. The ability to execute a single command in another subsystem with automatic return has been very useful.

Provision has been made for options that the user can control as he wishes for the amount of prompting, feedback, recognition mode, and for setting other user interface parameters whenever it seemed a standard interface might not be appropriate to some significant class of users.

A mechanism is implemented that enables the user, or someone acting in his behalf, to create a file stating what options he wants to run with. The system thereafter automatically sets these options when he enters. This facility can also be used with small extensions to subset commands. This user option capability, when coupled with the ease by which the user interface can be redefined using the Command Meta Language described below, makes possible tailoring the user interface to specific users or groups of users.

All operations that have a natural inverse command have been given one (although NLS still does not have an "undo" facility).[14] A general undo/redo facility has a number of technical difficulties and its value might be questioned. However, the ability to undo or redo the last one, two, or three commands would clearly be useful.

As indicated earlier the ability of the user to extend the system himself is important. There is a tradeoff between ease of extension specification and operational efficiency. In providing such a facility one does not have to be deeply concerned with efficiency if the task handled by the extension is performed infrequently. If the operation is performed frequently, then it should probably be inserted as a system feature and implemented efficiently by professionals. This area is ripe for much additional development. The extensions must be specified in some language to indicate what sequence of events is to take place, what arguments to collect, and so forth, when a given user action is performed.

NLS now offers two forms of extensibility. The first allows users with some basic programming knowledge to write programs in the Algol-like L10 language in which the system is implemented, calling NLS system primitives as needed. They can use the Command Meta Language to specify a user interface if desired.[12] These programs can be installed by the user as one of his default subsystems, loaded as subsystems as needed, or used as content analyzer patterns.[8]

The user can also write sequences of NLS commands and have these sequences executed at will. A specific sequence of commands can be automatically invoked when the user first enters NLS.

HELP, STATUS, AND PORTRAYAL FACILITIES

The user interface must implement a man/machine dialog. In this section, we discuss issues from machine to man. The discussion centers around the use of displays, with comments on how the problems are dealt with for typewriters. Let us examine some of the types of information that the user needs in order to keep his bearings.

There are three main areas or dimensions along which the user needs information to help him (a) to know where he has been, (b) to know where he is, and (c) to know where he can go from here. Clearly the command language and user interface must offer provisions to move in these spaces as well as obtain status.

(1) Information Space—The user needs to know where he is in his information space, and what view or portrayal of the many possible is being displayed to him. Generally he arrived at his present position from previous points and he may want to be able to return to previous points or views as well as to move on.

(2) Subsystem or Tool Space—In workshops containing many tools and commands, the user needs to know which tool or tools are active, which ones he was in previously and their order, and which ones he can enter from here.

(3) Command Syntax Space—During the specifications of a command, the user may need to know what he can or is expected to do next and how to back up to a previous point.

The NLS display screen is organized into windows, as described in some detail in Reference 9. These windows are arbitrary rectangles. Windows can be displayed essentially all the time or overlaid with others. Windows can grow dynamically. Some windows are allocated and displayed or not displayed under system control for status and feedback information. Others can be created and manipulated by the user for display of his information space. Items selected from the screen by pointing at them with the mouse are indicated with an appropriate feedback mark. With typewriter terminals, one does not have this two-dimensional random display capability. While the same information can be given to the user, less can be given automatically, or at least the information must be given in an altered form.

(1) Information Space—The present NLS information space is hierarchically organized. A user has a directory or directories within which there are files. A file can contain notes on many subjects stored under various headings, his mail, or single documents. Files in turn are hierarchically organized as a tree of information nodes containing text, graphics, or both.

Files can contain cross citations to specific points within other files or the same file, thus creating networks. NLS has appropriate commands for moving within and between files and for obtaining a display of the path over which one has traveled, and commands for backtracking along this path.[1]

Display screens have a limited number of lines within which to display information, and typewriters, even at 30 chars/sec or higher, cannot quickly and easily print out large documents. Also, the user often wants to see a summary or overview of a document or have it formatted in special ways to aid his understanding. To meet this need for easy control of information portrayal, NLS has a concept called "view specification." The user can change his "view" within the commands for moving in information space or by separate command. So that he can be reminded of his current view, the most commonly used view parameters are fed back to him in a small window in the upper right hand corner of the screen. When he is at a point in a command where it is permissible to change views, this fact is fed back both by prompt (if prompts are turned on) and by enlarging the characters in the view-feedback window. For more discussion on moving, viewing, and portrayal in NLS see References 1 and 4.

(2) Subsystem or Tool Space—NLS is viewed as a collection of tools (subsystems) that can be used cooperatively. Each subsystem contains a number of logically related commands and has a name, such as Base

(the collection of editing and file manipulating commands), Calculator, and so on. All the tools work on information in the same file structure and the user can move from one tool to another, or execute commands on a single command basis in any tool from any other tool, as mentioned earlier. The user can receive a display of subsystems available to him or an ordered list of the subsystems in which he has previously been.

The name of the current subsystem within which he is operating is fed back in a small window in the upper left-hand corner of the screen in DNLs and as a four-character prompt in TNLS.

(3) Command Syntax Space—Several levels of feedback and types of "help" are available to the user in formulating a command to the system. Each is described below. The Help database, to be described, clearly is also generally useful for understanding the system as a whole.

(a) Command-Word Recognition:
The options here were described earlier and this mode is primarily useful in minimizing keystrokes and in triggering additional feedback.

(b) Noise Words:
When the system recognizes a commandword or field, it generates what we call "noise words" set off in parentheses so the user can distinguish between what he has entered and what the system has added. For example, INSERT STATEMENT (to follow), MOVE WORD (from) aid the user to remember to designate a statement in the first case, or select the word to be moved. In the latter case, after selection, the characters (to) will be fed back to prompt for the destination of the move. The noise words aid the user in remembering what to do next. Novice users report that noise words are one of the most useful initial aids. As more experience is gained, the other aids take on more importance. This is an important point to note: users at different levels of experience value different forms of feedback. Usefulness is not only determined by the inherent characteristics of the aids, but also by how they are implemented.

(c) Prompts:
When the user completes the specification of a field in a command, he is prompted with some terse characters indicating the type of thing expected next and the alternatives available to him for specifying, selecting, or addressing the needed argument. In DNLS the prompts are displayed on the line below that used to feedback the state of the users command specification and appropriately positioned horizontally. In TNLS the prompts appear in the command specification feedback as appropriate. Users can turn prompts off, which some users of TNLS do when they reach a certain level of proficiency, although many highly skilled users always operate with them on. DNLS users tend to always operate with them on because the high speed of the display does not slow down work while providing useful information. Users can also specify terse prompting, in which case optional fields are not prompted for. Beginning users have indicated that prompting is useful, but would like prompts to be more mnemonic and of English type and word length.

(d) Next Options and Syntax:
If the noise words and prompts are not sufficient to jog a user's memory about what options are available to him next, he can strike a ? or a <Control-S>. If he strikes a ?, the system displays, in alphabetical order, all the command-words that are legitimate for the next field or more extensive information than is available in the prompts for other fields. If he strikes <Control-S>, the system prints out the syntax of the command from his present position to the end of the command. The ? facility is extensively used and is very useful in refreshing one's memory about infrequently used commands or new commands for a user with only a basic knowledge of command system concepts and vocabulary. The <Control-S> feature does not seem to be extensively used at present and may indicate that the ? facility is sufficient.

(e) Help Data Base:
If the above facilities are not sufficient because of uncertainty about a basic concept or vocabulary word or the user wishes more information about the effects or use of a command, he can enter the Help tool. Entry can be from the basic command level or from any point during command specification. In the latter case, the system utilizes the information input up to this point to take the user to an initial point that describes the specific command and field where he is located.[10]

Once in the Help Database, a simple set of command conventions and the organization of the database allow the user to easily examine related subjects or move to higher level descriptions.[10] There are many unanswered questions about the best structure of a help database, how to mesh online and offline documentation properly, and what forms of accessing mechanisms to provide for novices and skilled users. We are just beginning to review our experience with online help facilities to this point.

(f) Active Tutorial Help:
The next level of Help facility would be an active tutorial facility. We have not yet implemented such a facility but can see its value. An example of such a facility is the work going on at BBN on the NLS-Scholar system.[11]

ERROR MESSAGES AND RECOVERY

Error messages indicating an incorrectly spelled file name or improperly specified entity are fed back to the user in a window at the top of the screen. The user is left at an appropriate point within the command specification or, where necessary, he must start over again to respecify the command. The text of error messages is important and should be as specific to the problem as possible. This has implications within the system design for trapping error conditions as early as possible and determining the appropriate message for the specific error and total context of the user. While we have made progress in this area, there is much more that could be done to meet the need stated above.

There are now no automatic error correction mechanisms built into the system, such as spelling correction or "Do What I Mean" type facilities.[14] These would probably be useful to add when resources permit.

EDITING AND BACKUP DURING COMMAND SPECIFICATION

The user can perform certain simple editing and backup operations during command specification. At any point during command specification he can do a "command delete," which will take him back to the basic command level. This is useful if he gets confused and wants to return to a known state or changes his mind about which command to perform next.

The user can delete the last character input or last selection made on the screen with a "backspace character" keystroke or button push on the mouse. He can repeat this process and continue the incremental backup process to the basic command state.

He can also delete the last word input, or the field specified to date, with a "backspace-word" keystroke or button push on the mouse. He can also repeat this process backwards to the basic command state as well.

IMPLEMENTATION

The mechanisms and data bases needed to implement the user interface have been modularized and isolated as a "Frontend" that can run on a separate computer, such as a minicomputer close to the user, and communicate with the basic tool information processing routines ("Backend") over a communication network. The Frontend consists of terminal handling capabilities, a command language interpreter, and two data bases; a Grammar representing the language syntax and noise words; and a User Profile indicating how the user wants various parameters set for him, such as his prompt and command recognition modes, keyboard key translations, and so on. The Grammar is generated from a high-level description of the user interface written in a language special for this purpose we call Command Meta Language.[12]

Given this particular system organization, it is easy to tailor, subset, or modify the user interface for individuals or groups, or to create interfaces for new tools.

Furthermore all the levels of help information, except the Help Data Base, are derived from the Grammar, which guarantees their correctness as the system changes and is debugged. Various forms of hard copy documentation, such as command summaries, are also derived from the Grammar representation.

ACKNOWLEDGMENT

This work sponsored by the Defense Advanced Research Projects Agency and Air Force Systems Command's Rome Air Development Center, Griffis AFB, N.Y.

REFERENCES

1. Engelbart, D. C., and W. K. English, "A Research Center for Augmenting Human Intellect," *AFIPS Conference Proceedings*, 1968 FJCC, Vol. 33, pp. 395-410.
2. English, W. K., D. C. Engelbart and M. A. Berman "Display-selection techniques for text manipulation," *IEEE Transactions on Human Factors in Electronics*, Vol. HFE-8, No. 1, March 1967, pp. 5-15.
3. Engelbart, D. C., "Design Considerations for Knowledge Workshop Terminals," *AFIPS Conference Proceedings*, 1973 NCC, Vol. 42, pp. 221-227.
4. Engelbart, D. C., R. W. Watson and J. C. Norton, "Augmented Knowledge Workshop," *AFIPS Conference Proceedings*, 1973 NCC, Vol. 42, pp. 9-21.
5. White, J. E., "A High-Level Framework for Network-based Resource Sharing," *AFIPS Conference Proceedings*, 1976 NCC, Vol. 45.
6. Postel, J. B. and J. E. White, *Notes on a Distributed Programming System*, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, March 1975.
7. Roberts, L. G. and B. D. Wessler, *The ARPA Network*, Advanced Research Projects Agency, Information Processing Techniques Office, Washington, D.C., May 1971.
8. *L-10 Users' Guide: Content Analyzer*, Augmentation Research Center, Stanford Research Institute, Menlo Park, California.
9. Irby, C. H., "Display Techniques for Interactive Text Manipulation," *AFIPS Conference Proceedings*, 1974 NCC, pp. 247-255.
10. Lehtman, H. G. and K. Kelley, et al., *Query/help Software and Data Bases*, Knowledge Workshop Development Final Report RADC-TR-75-304, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, June 1974.
11. Grignetti, M. C., C. Hausmann and L. Gould, "An Intelligent" Online Assistant and Tutor—NLS Scholar," *AFIPS Conference Proceedings*, 1975 NCC, pp. 775-781.
12. Irby, C. H., "The Command Meta Language System," private communication.
13. Bobrow, D., et al., "TENEX, A Paged Time Sharing System for the PDP-10," *Commun. ACM*, Vol. 15, pp. 135-143, March 1972.
14. Teitelman, W., et al., *INTERLISP Reference Manual*, Bolt Beranek and Newman Inc. and Xerox Corp., 1974.

Terminal transparent display language (TTDL)

by CARL E. KREBS, C. BUMGARDNER and T. NORTHWOOD
INCO, Inc.
McLean, Virginia

ABSTRACT

Terminal Transparent Display Language (TTDL) is a software language that implements complete and effective communications in an on-line computer system containing two or more different types of terminals. TTDL integrates differing terminal display techniques into a common language, freeing the application programmer from the specifics of terminal data handling and allowing him to concentrate on his primary function of providing a service to the terminal user.

INTRODUCTION

INCO, INC. developed and is currently implementing Terminal Transparent Display Language (TTDL) as part of the Terminal Oriented Support System (TOSS) Research and Development project for the Rome Air Development Center (RADC). The purpose of TTDL is to provide a software language and support system which will allow communications to different terminal types without reprogramming. See Figure 1. Under TTDL, terminal differences become "transparent" and do not require consideration by the programmer.

TTDL was specifically designed for use on Digital Equipment Corporation (DEC) PDP-11 series minicomputers that are terminal-oriented. TTDL provides the capability to operate any number of different terminal types on this system because it approaches terminal transparency from a functional standpoint. TTDL was designed to support the logical functions required by the intelligence analyst, rather than the specific terminal devices he might wish to use. For example, TTDL allows the programmer to specify an "accentuation level" to emphasize a particular section of text. This level is then mapped into a specific terminal's physical characteristic; this may be blinking on one terminal, but may be reverse video on another terminal.

This paper addresses four specific areas of the TTDL system. Firstly, it describes the Display Specification Language, the primary communication tool of the application programmer.

Secondly, it describes the seven most important functional features of TTDL. They are:

- Screen Area
- Command Line
- Select Menu
- Screen Formatting
- Data Checking
- Control Input
- Display Library

Thirdly, it describes the separate software modules which drive the system. And lastly, it details the control and flow of data through the system.

By describing TTDL in terms of its user functions, this paper hopes to present a more meaningful software description; one from which the reader can understand TTDL's operation as well as its theory.

DISPLAY SPECIFICATION LANGUAGE (DSL)

TTDL defines displays using the Display Specification Language (DSL). This language is designed for use with either FORTRAN or MACRO-11 (PDP-11 assembly language). DSL is clear and concise and allows great flexibility in designing displays. Displays can be defined in three ways: statically, not changing during the program; dynamically, using data from the execution of the program; or a combination of both ways. DSL provides for naming a field or group of fields so that they can be referenced by routines which modify, extract data from, or dynamically accentuate such named fields. This accentuation can be applied to any or all fields in the display, either when the display is defined or later during program execution.

There are 16 levels of accentuation, including the normal level (no accentuation), and a "zero" or blank level (data input is not displayed). The blank level is useful for entering passwords or other classified information, and for entering error messages into the display, which can then be used merely by dynamically changing the accentuation level.

DSL provides formatting capabilities, which include:

- Floating tab positions (up to 16) which are set

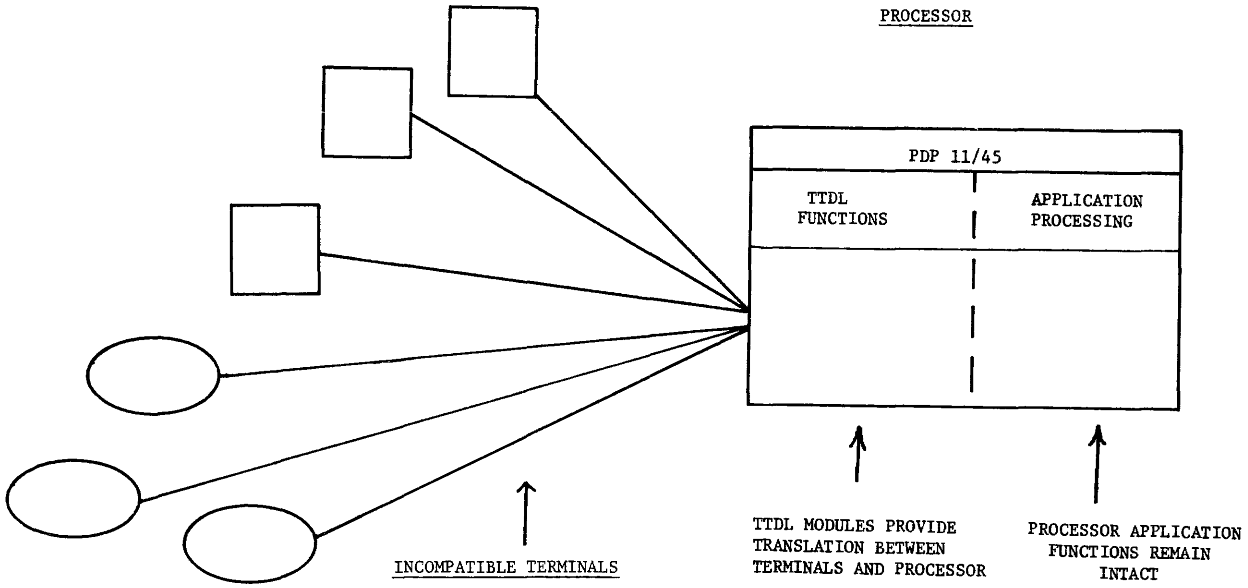

PROCESSOR
PDP 11/45
TTDL FUNCTIONS
APPLICATION PROCESSING
INCOMPATIBLE TERMINALS
TTDL MODULES PROVIDE TRANSLATION BETWEEN TERMINALS AND PROCESSOR
PROCESSOR APPLICATION FUNCTIONS REMAIN INTACT

Figure 1—TTDL application.

according to actual data length for the fields and the width of the target screen display area.

- Right and left justification around a tab position.
- Center text on a line.
- End page.
- New line.
- Justify text against the right margin.
- Fill the space between two fields with dots.

DSL allows the programmer to specify data attributes for input fields, such as data type and length. Data types include:

- String fields which consist of any character in the print set. Specific string attributes include alphabetic, numeric, blank, and special characters (all other printing characters). Any combination of the four is also allowed.
- Integer fields.
- Real fields in which scientific notation is also supported.
- Minimum-maximum field where the integer entered must be between two limits.
- Two character choice fields consisting of one position where the user must enter one of two specified characters.

Any of the above fields may also be specified as mandatorily requiring input as opposed to optionally requiring input. Input fields can also be accentuated.

DSL thus frees the programmer from varying considerations of screen size, display techniques, accentuation techniques, input methods, and required hardware protocol.

FUNCTIONAL FEATURES OF TTDL

Screen areas

The terminal display device can be divided into multiple independent regions, each known as a Screen Area (SA). Each SA can be treated as a separate display, allowing multiple programs to be run simultaneously on a single terminal. SA's can be defined in any realistic size.

Command line

Each terminal has a one-line command line. Input on this line is passed to a command interpreter which checks the command against a list of system commands. The command is then dispatched to the appropriate routine for implementation. Error messages, either syntax errors from the interpreter or execution errors from the service modules, are displayed on the command line at the end of the entered command. Current commands include running and aborting tasks, and display paging commands.

Select menu

A common programming technique is to provide the user with a list of choices, or selection menu, for the user to select from. TTDL provides a select construct in which the programmer merely lists all the choices in the menu. The prompt line (e.g., ENTER DATA

. . .) is automatically generated by TTDL, a different prompt being available for each terminal type. This construct also allows the terminal user to make full use of any available input devices which the terminal supports. These may include light-pen, cross-hairs, function keys, or he may simply type in his choice. The choice made is echoed in the input field of the prompt line. The program sees only an integer denoting the choice selected.

Screen formatting

TTDL provides automatic display formatting by fitting the logical display to the target SA. The process includes breaking lines that are too long by backscanning for a blank and breaking at that point for a display-only field, or breaking exactly at the end of the line for an input field. The display is broken into pages if it cannot be made to fit on a single page. Efforts are made to fit the display on one page, such as ignoring "new line" codes for compression of selection menus. If the display must be paged, efforts are made to keep a logically connected field from being broken between pages by moving the entire field to the next page. Page information is also included at the bottom of each page, i.e., page number, whether there are more pages or not, and paging commands which can be entered by input devices—if they are available.

Data checking

TTDL provides automatic input editing to insure that data meets the restrictions assigned by the programmer. The user is not allowed to leave the current page until all errors are corrected. The number of errors is displayed on the bottom line of the SA. Fields requiring mandatory input must also be filled in correctly, although optional fields can be left blank. The fields are then marked to show the application program those that had data entered.

In the case of string fields, any spaces not filled in always appear as nulls to the programmer, no matter what character is used to reduce the limit of input fields on the terminal. Selection menu choices are checked for the validity of the total number of choices available.

Control input

Control sequences can be entered apart from normal data input. These sequences trigger control functions on the SA in which the programmer is working. These functions include:

- Command line—transfer to enter data on, and return from command line.
- Paging commands—page forward, page back, first page.
- Enter data—data is ready to be returned to the application program.
- Cursor control—move cursor up, down, right, left, home. Cursor only stops on unprotected areas.
- Tab—tab to next or prior input field, tab to next field containing incorrect data (flagged by data checking routines), tab to the first input field on the next line.
- Move to the next active SA.
- Erase input field.

Each function is declared valid or invalid and ignored pursuant to the status of the SA.

Display library

The user can store displays on disk through a display library capability, referring to them by a six-character name. Functions supported are store, retrieve, and delete. This capability frees the application program from the storage overhead of having the displays defined internally. Checks are made for duplicate names when storing. When retrieving displays, the actual display name is checked against the active displays; if it has the same name, it is renamed, and the new name is given to the program to use. This allows the same display to be used concurrently by different terminals controlled by the same program.

TTDL SOFTWARE MODULES

TTDL was designed using two kinds of software modules: primary modules, which drive the system; and support modules, which provides sub-processing and data support for the primary modules.

The primary modules are the Preprocessor, Postprocessor, Secondary Input/Output, and Primary Input/Output modules. They handle the actual transfer of data to and from the terminal.

- The preprocessor translates the DSL into a Terminal Independent Format (TIF).
- The postprocessor formats the data to fit the SA, and provides dynamic display paging support.
- The secondary input/output module translates common capabilities into terminal dependent functions.
- The primary input/output module transfers data to and from the terminal.

The support modules are Buffer Manager, Process Control, Field Routines, Application Interface, and Display Library modules.

- The buffer manager directs the storage and retrieval of TIF data.
- The process control module supports intra-task and inter-task communications.

- Field routines allow modification to and retrieval of data from the display by the application program.
- The application interface module provides a table-driven interface between the applications program and TTDL.
- The display library module handles the storage and retrieval of displays from disk.

The following paragraphs provide a complete description of each TTDL software module.

PRIMARY TTDL MODULES

Preprocessor (PRP)

The preprocessor is a finite-state compiler which translates the DSL specified by the application program into a Terminal Independent Format (TIF). Using state tables, the preprocessor translates the data from DSL to TIF, which is suitable for storage on disk and recall for retrieval on any type of terminal. The preprocessor is completely terminal independent, requiring no information about the display terminal.

As a finite-state compiler, the preprocessor uses a series of macros to conduct state-translation processing. This construction allows easy additions to the language by merely making the additions to the state tables and combining them with the appropriate processing macros.

Postprocessor (PSP)

The postprocessor formats the TIF produced by the preprocessor to fit the target screen area. It builds a transaction file correlating data in the TIF to locations on the SA. Algorithms break lines that are too long, and automatically pages displays which are too big for one SA. The resulting transaction file is sent to the secondary I/O module for further processing. The postprocessor is virtually terminal independent, requiring only SA size and the number of display characters required for certain functions.

Secondary I/O (SIO)

The secondary I/O module produces the actual data stream sent to the terminal. As input, it uses the TIF, the transaction file built by the postprocessor, and a set of terminal dependent protocol tables and routines.

The secondary I/O is table-driven, a feature which facilitates addition of new terminals to the system. Addition of a new terminal requires only the addition of protocol tables and those routines which are unique to the terminal or new to the system. As more terminals are added to the system, the number of new routines decreases; most routines are common to a class of terminals.

Specific functional capabilities of SIO include: adherence to hardware protocol for each terminal; display field accentuation; addition of dots for the DSL feature; compensation for differences between terminal types; and hardware simulation with software routines.

Primary I/O (PIO)

The primary I/O module handles the actual transfer of data to and from the terminal. Inputs to primary I/O from the SIO are the address of the data and its length. Important features include: interruption of a read to allow a write to occur, and restoration of the read at the exact point of interruption; queueing of other reads while a read or write is in progress; validation of control sequences entered by the user—invalid sequences are ignored; support of zero level accentuation by echoing blanks to the terminal when data is entered; and asynchronous operation, by processing requests on other terminals until recently issued I/O is completed.

The primary I/O is also table driven, having tables of write routines, read routines, etc., which are dispatched by terminal type.

SUPPORT MODULES

Buffer manager

TTDL uses dynamic storage for its work areas, the majority of which are used for TIF. The buffer manager is responsible for handling of these buffers.

TTDL uses 512 byte blocks of dynamic storage allocated by the Intertask Coordination Module (ICM) of TOSS. Using a logical-to-physical mapping algorithm, the buffer manager translates TIF addresses into a specific block of TIF. The buffer manager pages the buffers to disk on a demand basis when in-core storage runs short, using an extensive algorithm to determine which blocks to write out. A currency concept is also used, through which a certain buffer is kept current for the TIF being processed, with a pointer to the current character in the buffer. Functions include: set currency; get current character; put a character; access a block; deaccess current block; and map into current location. This last function allows the requesting module to directly access the current block.

Process control module

The process control module provides the control required by TTDL to permit the simultaneous support of multiple tasks, multiple terminals and terminal types, and multiple screen areas on each terminal. The process control module assigns a Process Control Block (PCB) to each individual process. The PCB contains all appropriate data for the process, including a stack area where data may be stored if the process is suspended. In addition to providing intra-task control, inter-task communication is achieved by queueing PCB's from one task to another.

The process control module also resolves resource contention by suspending a process until the required resource (such as nodes, buffers, or PCB's) is available.

Process control functions include: passing control to another process; suspending a process; spawning a new process; ending a process; and subprocess calling and return (analogous to subroutine call and return). These functions manipulate queues of process. A dispatcher module provides the control mechanism for dispatching processes to be run and identifying the task in which the process resides. The dispatcher also handles Service Request Blocks (SRB) which are requests for service issued by application programs through ICM.

Field routines

Field routines consist of a group of modules which allow the application programs to interact with the TIF for their display. Individual modules include: initialize a field to the null state; set fields to string, integer, or real values depending on field type; retrieve data from string, integer, or real fields; change accentuation levels of display fields; test the availability of input for input fields; and set origins for generated field names. The largest modular group is a set of routines to find specified fields in the TIF which the application programs desire to access. Since the processing performed by most routines is minimal, addition of new routines is easily accomplished.

Application interface module (AIM)

The application interface module is the entry point for all application calls. It checks the format of each call and formats a service request block for transmission to the appropriate TTDL task. The AIM is FORTRAN compatible, and addition of a new host language would require only a small system enhancement. TTDL task specifications would not be affected.

The AIM is table driven, so that a new call requires only the addition of two macros. The AIM also provides conversion functions for dynamic display definition from data provided during execution.

Display library

The display library module consists of two parts. One part performs the directory functions and allocation of disk storage. The second part, the external processor, performs the formatting of the display TIF and associated information for storage on disk.

A directory entry consists of a 6-character ASCII name, the virtual block number of the display, and the number of blocks allocated. Each directory has a header which contains a pointer to the next directory and the number of blocks allocated for a high block number. Each directory is one disk block long. A deleted directory entry is marked such that the blocks may be reused.

While the display library keeps track of display storage, the external processor formats the information in the TIF and additional information required when the display is reformatted. Parameters to the external processor include a buffer to store the information in and its length. The external processor can also be used to send displays to and from other systems.

DATA AND CONTROL FLOW

The following paragraphs describe the typical flow of data and processing to and from a terminal. This flow is illustrated in Figures 2 and 3.

Display definition and output

Operations start with a call to the routine DISDEF (display definition). DISDEF formats a request specifying the argument and passes it to the preprocessor. The preprocessor then maps into the application program and retrieves the DSL. The DSL is then converted into TIF and given to the buffer manager for storage.

Once the display is defined (by one or more calls to DISDEF) the program indicates its desire to output the display via the FLASH call. This is dispatched to the postprocessor which creates a transaction file based on the TIF and the size of the target SA. This transaction file relates TIF data to SA position.

Control is then passed to the secondary I/O module, which uses the TIF, the transaction file, and terminal dependent tables and routines to create the output stream. SIO also adds the hardware protocol for such features as protected/unprotected fields, emphasis (blinking, reverse video, etc.), code transliteration, and cursor positioning.

Finally the buffer containing the output stream is passed to the primary I/O module, which transmits it to the appropriate terminal.

Data input

After the display has been output, the SIO then prepares to input data. This data can consist of control sequences or input for the program. The SIO sends a request to the PIO for each field, stating how many

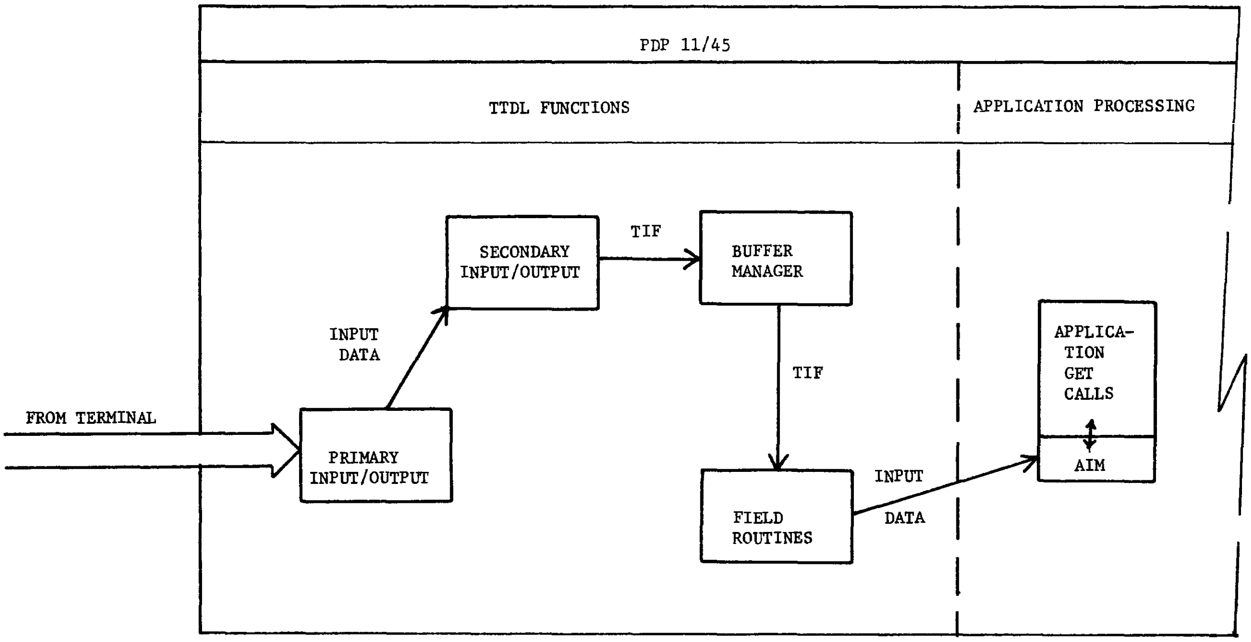


Figure 2—Input flow.

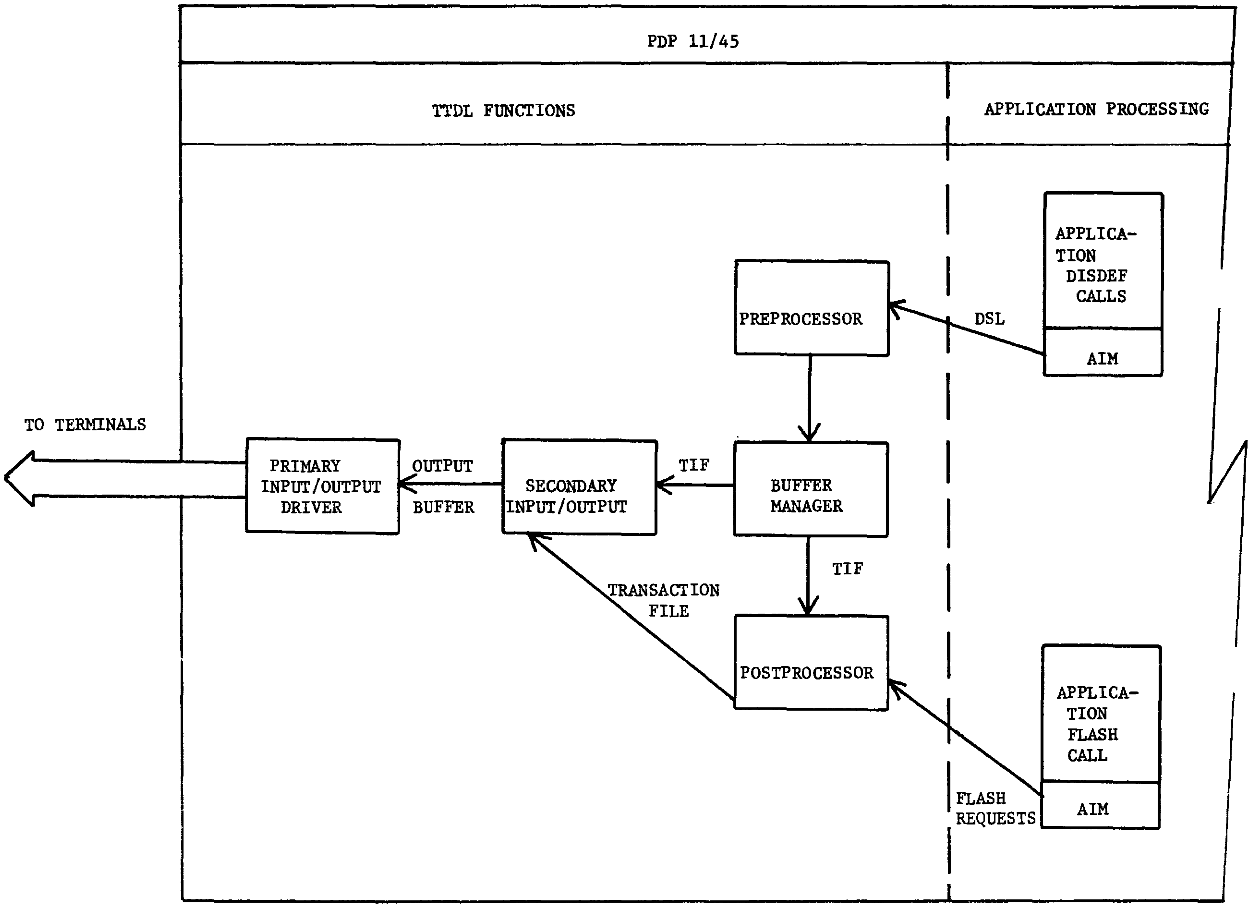


Figure 3—Output flow.

characters are required, any accentuation information, and buffer addresses for the input and output buffers. The PIO then returns control whenever the number of control characters or a valid control sequence is entered. The SIO then either goes to the next field, or processes the control sequence. When all the data is entered, the SIO returns control to the application program, which can then modify or retrieve the data entered.

SUMMARY

As can be seen from the above discussion, TTDL is a useful flexible language which can effectively facilitate communications among the many and various terminals of large data processing systems. Such a language delivers powerful capabilities to system users—capabilities which are necessary in the rapidly changing data processing environment.

Working set restoration—A method to increase the performance of multilevel storage hierarchies

by PETER SCHNEIDER
Siemens AG
Munich, Germany

ABSTRACT

The emergence of new storage technologies such as Charge Coupled Devices (CCD) and Bubbles with access times which lie in the access gap between semiconductor memories and rotating magnetic storage media is another important step toward implementing multilevel storage hierarchies.

However, a comparison between a three-level storage system, consisting of cache, page buffer and CCD main memory, and a conventional two-level main memory system will show that the three-level hierarchy using the transfer on demand strategy has an effective access time which is higher by about a factor of 2.

Yet, through better use of the program locality the access time of a three-level system can be reduced to that of the two-level cache/page buffer system. Using this method, the so-called working set restoration, the working set of pages of the next program to be run is loaded into the page buffer during execution of the active program. The required page transfer operations are executed concealed and are thus not time-critical for the processor. This means that for program processing only the access time to the two-level system becomes apparent.

The advantage of a three-level system of this type lies not so much in the improved performance but rather in the lower costs, since it permits the use of a large-capacity main memory on a technology level which is cheaper by a factor of 2 to 4 as compared with MOS RAM.

INTRODUCTION

In the planning of large computer systems, there is a pronounced trend toward increased use of multilevel storage hierarchies. The first step in this direction was the development of the software-controlled virtual memory.

The emergence of the various semiconductor technologies such as the bipolar and MOS technologies, which brought about random access memories of various densities, speeds and costs, led to the development of directly addressable two-level main memory structures. These consist of a cache designed in fast and hence expensive bipolar technology and a main memory designed in cheaper, albeit slower, MOS technology; see Figure 1. By making use of the locality of the active programs, the effective access time of the main memory system can be reduced to nearly that of the cache. The introduction of a two level main memory system in machines with a virtual memory was the next step toward a system enhancement through the use of storage hierarchies. Recent developments include new storage technologies such as Charge Coupled Devices (CCD) and Magnetic Bubble Domain Devices (MBD) of which some were announced as products. Their access times lie in the long-existing access gap between the semiconductor technologies used in the main memory and those of the secondary storage media. The per-bit costs are also estimated to fall in between the price categories of the two known technologies.

Because the new storage technologies have an operation mode different from that of the random access storages, they are called Block Access Memories (BAM).[1] Whereas with RAMs any bit can be addressed within an equally short time, BAMs require long access times for single bits while, for a sequence of bits, the longer access time occurs only at the beginning with the remaining bits following sequentially at a high data rate.

In the system considerations outlined in the following paragraphs, the CCD technology will be used as an example representative of the above-mentioned new technologies.

CCD TECHNOLOGY—ITS COST AND PERFORMANCE

The operation of Charge Coupled Devices (CCD) is founded on the basic concept of storing information in MOS capacitors in the form of charge packets and to

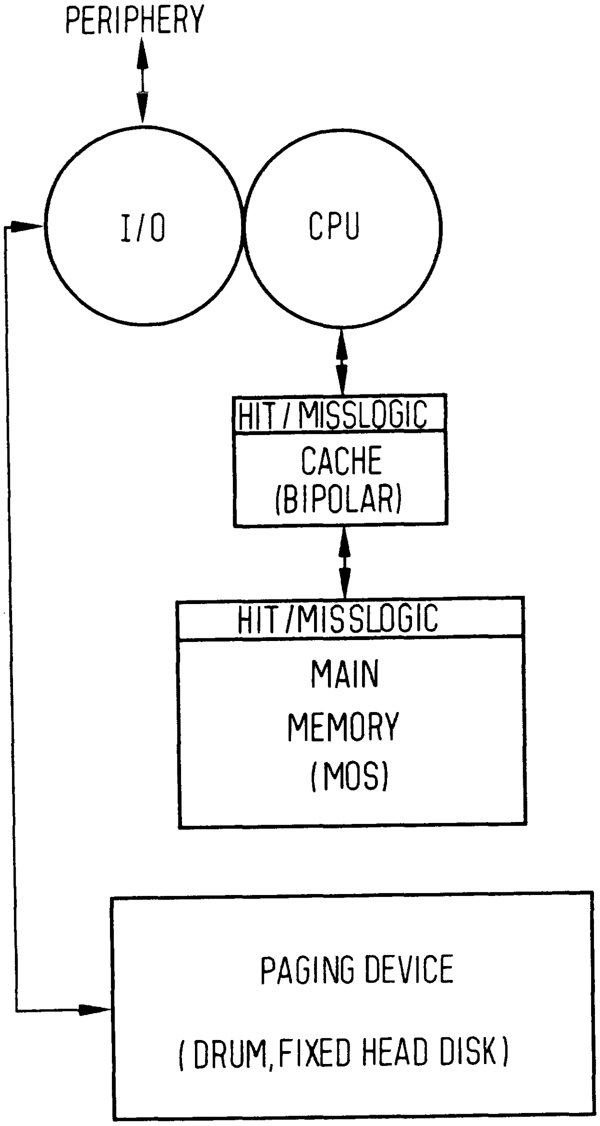


Figure 1—Two level working memory hierarchy with paging device

shift it at the semiconductor surface from one capacitor to the next. This technology is thus suitable for generating shift registers for the storage of analog as well as digital information. The inherent advantages of the CCD technology are that it is based on the tried MOS technology, storage locations can be extremely small and a high production yield is expected. As compared with MOS RAMs, memory densities greater by a factor of 2 to 4 and per-bit costs lower by a corresponding factor are anticipated for the CCD memory chips.

Memory chips implemented in this technology contain randomly addressable shift registers which are closed via read/write terminals. Since, in the CCD technology as well as in the conventional MOS technology, information is stored dynamically, a refresh of the entire information written is needed after several milliseconds. This is also done by the read/write terminal. Due to shift-frequency-dependent transfer losses, which occur when the information is shifted from one capacitor to the next, the length of shift register loops is limited. As an example, lengths of up to 256 bits are believed to be realizable for a shift frequency of 10 MHz.[3]

For the following system considerations, devices with a capacity of 65,536 bits, a setup of 256 loops with 256 bits each, and a shift frequency of 5 MHz are assumed. The time required for a complete pass of the information stored in a loop is then 51.2 μs; the mean value for accessing an arbitrarily chosen bit is after one half pass, corresponding to 25.6 μs. Memory chips incorporating these characteristics will be available in the near future.

Some system-engineers believe that these new technologies will challenge only drum and fixed-head disk storages, and thus solely consider the possibility that these technologies replace the conventional paging device in the virtual storage system. By contrast, a discussion in Reference 4 is based on the assumption that the use of a CCD main memory within a three-level directly addressable main memory might obviate the need for a paging device.

However, since the CCD technology, viewed price-wise, can probably compete with MOS RAMs but not with secondary-memory technologies, the following question arises: Can costs be cut maintaining the same system performance if a two-level main memory is replaced by a three-level main memory structure with a CCD main memory, assuming that the virtual storage concept with a conventional paging device is retained?

VIRTUAL STORAGE SYSTEM WITH THREE-LEVEL MAIN MEMORY

To clarify this question, we must first look at the functions performed by today's main memory.

To ensure efficient utilization of the central unit, it is necessary to keep the current pages, the co-called working set[5] of several programs in the main memory even if processing of these programs is interrupted due to secondary storage accesses. If, for instance, program pages are missing which have to be fetched from the paging device and entered in the main memory, processing of the active program has to be interrupted. The page entry takes several milliseconds during which time the central processor would have to wait with nothing to do if no other executable programs were available.

However, if several processes are in main memory, the central unit moves on to another process if the active process has to be displaced. Among the processes kept in main memory are those waiting for a page to be entered from the paging device. This ensures that, upon reactivation, processes will not have to be deactivated because pages are missing which were overwritten just before.

This holding of working sets of several programs plus the necessity of also having some operating system routines reside in main memory already require main memory capacities of 1 MByte or more for today's timesharing or multiprogramming environments. Due to the mounting storage requirements of individual users as well as the increased use of the installations for multiprogramming applications, an increased memory capacity of over 16 MByte will be needed.

This very fact presents a strong argument in favor of using a cheaper memory technology in the main memory. However, measures must be taken to guard against a loss of system performance. Due to the CCD memory access time which, compared with random access working memory, is longer by several orders of magnitude, an outright replacement of the main memory is not possible because resident operating system routines must remain readily accessible and the time of access to data of active programs must likewise not be adversely affected.

Considerable savings can, however, be achieved in terms of expensive random access storage capacity by dividing the main memory capacity into two categories using the following criteria:

(a) A large capacity memory, which will continue to be called main memory and is designed in CCD technology, for storing non-active working sets;

(b) a small-capacity random access memory, the so-called page buffer containing the currently active process as well as the resident and a small number of exchangeable operating system routines.

A strategy equivalent to the virtual memory's paging on demand might also be used here as a load strategy between main memory and page buffer. Pages could then be transferred to the page buffer each time they have been found missing by a built-in hit/miss logic. This method will be referred to as transfer on demand.

This division of the main memory results in a three-level main memory system consisting of cache, page buffer and main memory; see Figure 2. Originally, this division was made under the aspect of cost advantages alone, but we have to examine now whether a procedure of this kind will not result in a loss of performance vis-a-vis a conventional system.

TWO-LEVEL AND THREE-LEVEL MAIN MEMORY SYSTEM PERFORMANCE COMPARISON

For a comparison of the performance rate of different main memory systems, we use the effective access time for read operations

$$T_{eff}=h_1 \cdot t_1+(1\text{-}h_1)\cdot[h_2\cdot t_2+(1\text{-}h_2)\cdot t_3]$$

since only read operations are time-critical whereas write operations can in general be run concealed while the processor is busy. h_1 and h_2 are the hit probabilities for read accesses, t_1, t_2, t_3 the access times to the different hierarchy levels.

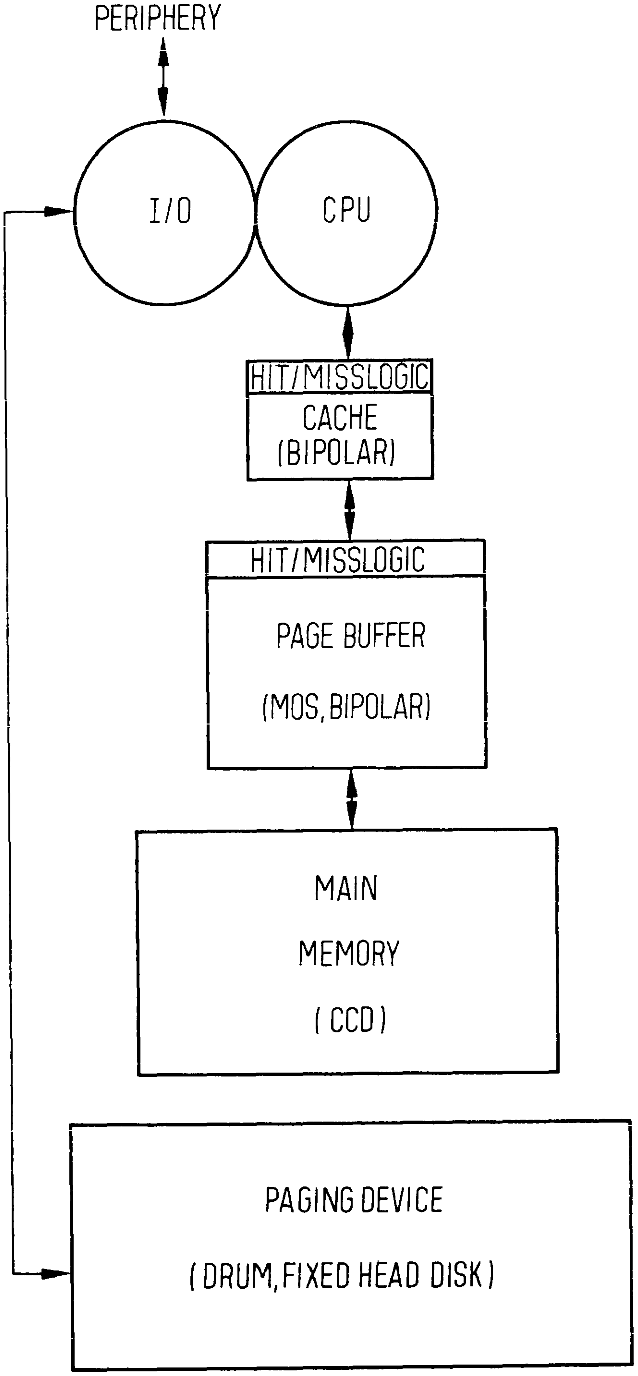


Figure 2—Three level working memory hierarchy with paging device

Let us first consider the two-level memory hierarchy cache/MOS main memory. Assuming

$$h_1=95\%, t_1=100 \text{ ns}, h_2=100\%, t_2=1000 \text{ ns},$$

the effective access time T_{eff} will be 145 ns. By comparison, a longer mean access time of 250 ns results for the three-level hierarchy with CCD main memory, page buffer and cache, when $h_2=95\%$ and $t_3=51.2$ μs.

Since the time of access to the CCD memory (t_3) is much greater than that of the page buffer (t_2), the effective access time of the overall memory is extremely susceptible to changes in the page buffer miss rate. If this increases heavily, the effective access time assumes values that are too high to be tolerated.

These considerations lead to the tentative conclusion that under adverse conditions the performance of the three-level hierarchy will be worse than that of its two-level counterpart when using the transfer-on-demand strategy assumed here, which works on the principle that pages are fetched from the main memory only after a page miss has occurred. In order to allow the cost-saving three-level hierarchy to be used without degradation, some means must be found to either prevent or at least significantly reduce page misses, which are responsible for longer access times. This may be achieved by use of the working set restoration method.

WORKING SET RESTORATION

This term will be used to designate a method designed to provide a program, upon its reactivation, with the specific memory environment it requires for optimum processing. This is where the locality of a program, referred to as working set of pages,[5] comes in. Under the conventional method, deactivated programs are, upon their reactivation, provided with that number of page frames for entering the pages which corresponds to the actually required number of pages during the preceding processing interval. In contrast to this, the method proposed here attempts to retrieve the actual pages addressed in the preceding interval rather than mere page frames.

A similar method is discussed by Tung[6] who suggests that shorter transfer times between two memory hierarchy levels be achieved by interleaving the main memory in as many ways as there are page frames in the page buffer, i.e., that the number of main memory modules should match the number of page frames in the page buffer. The interface width is equated with the page size. Nevertheless, there are processor wait times under this system during process changeover until a new process can be started. Besides, this concept would require a main memory with an extremely large capacity and the use of very wide interfaces would also pose a problem. Further, it should prove difficult to ensure an optimum distribution of all active pages over all possible memory modules—a necessity if the method proposed by Tung is to work properly. For these reasons, implementation of such a system cannot be advocated.

Since it does not make any difference in the time total whether all pages are transferred consecutively or individually upon request, it is suggested here to load the working set of a program concurrently with the processing of another process so that it does not occur time-critically for that process. To allow this, the page buffer capacity must be expanded by an additional area so that one area will always be available for program processing and another for loading the working set of the successor process. Further, the time available for concealed loading must be sufficient. However, it can be safely assumed that it is, because it is made up of the runtime section of the active program and the operating system execution time for the subsequent process change.

Through the use of this procedure, the access time of the three-level system is reduced to the access time of the two-level cache/page buffer system since the majority of page misses were forestalled in a non-time-critical manner. The third level will have to be accessed only when pages which reside in the main memory but not in the page buffer are added to the working set of the active program.

SYSTEM CONFIGURATION

Having explored the theories and facts suggesting that the use of the working set restoration concept as a load strategy between page buffer and main memory seems practical, let us proceed to discuss a sample system configuration and its management.

The memory system, Figure 3, has a cache with generally known characteristics and of conventional size at the top level. Since the cache is independent of the load strategy used between second and third level, it is excluded from the following discussions.

The 512 KByte (K=1024) page buffer is divided into four modules: Modules 1 and 2 are designated for program processing or for loading the successor process working set; Module 3 contains resident operating system routines; Module 4 contains various routines of the exchangeable operating system.

Each of these modules has a capacity of 128 KByte. At any given time, the active program is present in only one of the two modules slated for processing. Both modules contain the page buffer addresses from Ø to 31 (page size 4 KByte). This ensures that, during processing of a program in, say, module 1, the successor process can be entered in module 2 without affecting program execution. Besides this, the allocation of main memory and page buffer addresses is simplified in so far as all main memory addresses are always allocated to page buffer addresses 0 through 31.

The page size of 4 KByte is identical to that used in the virtual memory system. The storage technology employed in the page buffer must be able to accommo-

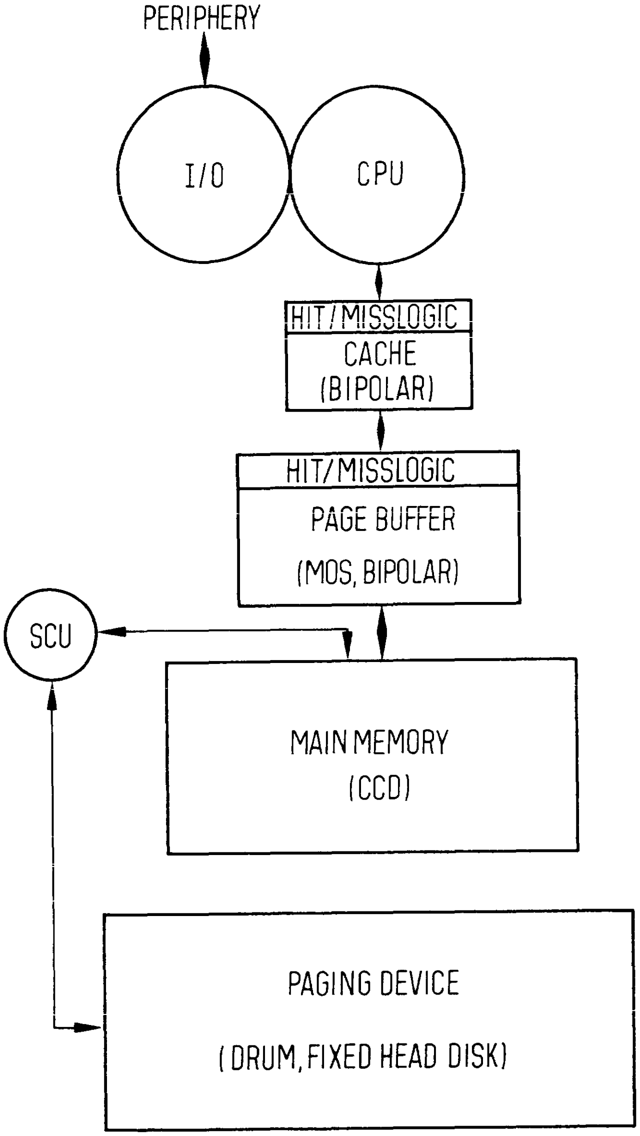


Figure 3—Scheme of the working memory system with direct paging between CCD-main memory and paging device

date the data rate transferred from the CCD memory over a data path with a width of, say, 8 Bytes.

A CCD memory of modular design is assumed as main memory: 144 CCD memory devices with a capacity of 65,536 bits contribute one bit each to a 16-Byte word which is transferred via the 8-Byte interface using the two-way streaming method. Consecutive words of a 4 K-Byte page are successively stored in the CCD loops; thus a page can be read out or written in 51.2 μs with one parallel cycle of all 144 loops. This corresponds to a data rate of 80 MByte/s. The module comprises 256 pages, i.e., it has a total capacity of 1 MByte, Figure 4. Depending on the desired system performance, the CCD main memory can be implemented with capacities from 1 MByte to 16 MByte or more with the aid of these modules.

A hit/miss logic implemented in the form of an associative memory is visualized as a means of checking whether the pages requested by the processor are entered in the page buffer. The associative memory always describes the specific page buffer module (1 or 2) containing the program (addresses 0 through 31), and module 4 containing the operating system (addresses 32 through 63). This ensures that the entries, by virtue of their being located in the hit/miss logic as they are, point to the page buffer address. Another section of the hit/miss logic—again with addresses 0 through 31—will be used for storing the working set table of the successor process. In addition to the main memory address entries, there will be a write bit which indicates whether a page entered in the page buffer was modified and therefore does no longer match the original in the main memory.

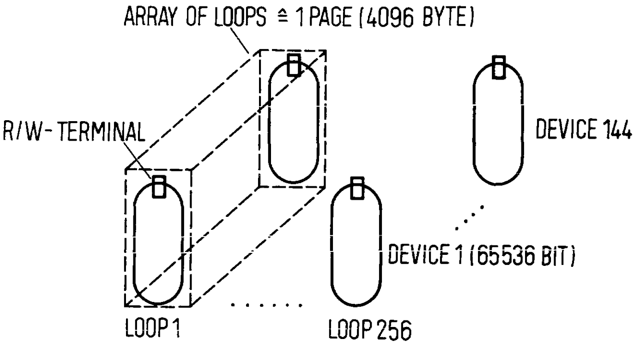


Figure 4—Scheme of a 1 MByte CCD-memory module

In the system discussed here (cf. Figure 3), the exchange of pages between main memory and paging device is not performed via the central processor as in so many existing systems, but rather via a storage processor (SCU) assigned to the memory. This processor has its own memory for storing the pages to be exchanged between main memory and page buffer until they can be entered without interfering with the processes taking place between main memory and page buffer. In addition, the storage processor is responsible for loading the working set of the successor process.

After this overview of the system's hardware components, let us now turn to the specific software requirements for working set restoration.

Each process will have a table assigned to it describing these pages (up to 32) that were entered in the page buffer by that process during the most recent processing period. These tables are stored in the memory of the storage processor according to their internal process code numbers and can be directly addressed by means of these numbers.

In the order to start the working set restoration procedure, the process requiring loading operations must be known to the storage processor. This information is extracted from the process queue maintained by the operating system and is passed by the central processor to the storage processor as soon as this interrupts a program and changes over to another one. This information enables the storage processor to control working set restoration proceedings.

FUNCTIONS

We will now discuss page miss handling and activities involved in working set restoration, Figure 5.

Page miss

After address translation in the central processor, a request address is sent to the memory system. In most cases, an access can already be satisfied in the cache. If a cache miss occurs, the address is switched through to the page buffer. The hit/miss logic then determines whether the requested address is present in the page bŭffer. If the desired page is not available there, it must be fetched from main memory. This is done by switching the address through to the main memory, after which page transfer is started. A processor-requested word within a page is switched through to the processor during page entry and, on the basis of the foregoing, will be received after a mean time of 25.6 μs.

In addition to pages which were read only, the page buffer also contains pages which were modified. Since these pages no longer match their original in the main memory, they must be written back to main memory before being overwritten. If both the requested page and the page to be written back are contained in the same module, this may require two page transfer times. In order to avoid such displacement procedures with page miss, one buffer page is always reserved into which the requested page can be loaded immediately. If the page released for overwriting by the replacement algorithm was modified, it is subsequently written back in non-time-critical fashion to main memory, and then becomes a new buffer page. Thus, 32 pages are active in the page buffer at all times. Yet, since one of these pages is being marked free, a page entry due to page miss can be handled in the shortest possible time.

Working set restoration

A working set is to be transferred from the CCD main memory to the page buffer whenever an active process is displaced. The central processor immediately switches over to the next process while the storage processor starts loading the working set of the next executable process; see Figure 5. The storage processor first reads out the 32 hit/miss logic entries describing the page buffer contents of the program just displaced, and updates the associated page table. In the course of this, it determines which pages have to be displaced from the page buffer and simultaneously checks to see, by way of a comparison with the successor process page table, whether any successor process pages fall into other modules than the pages to be displaced.

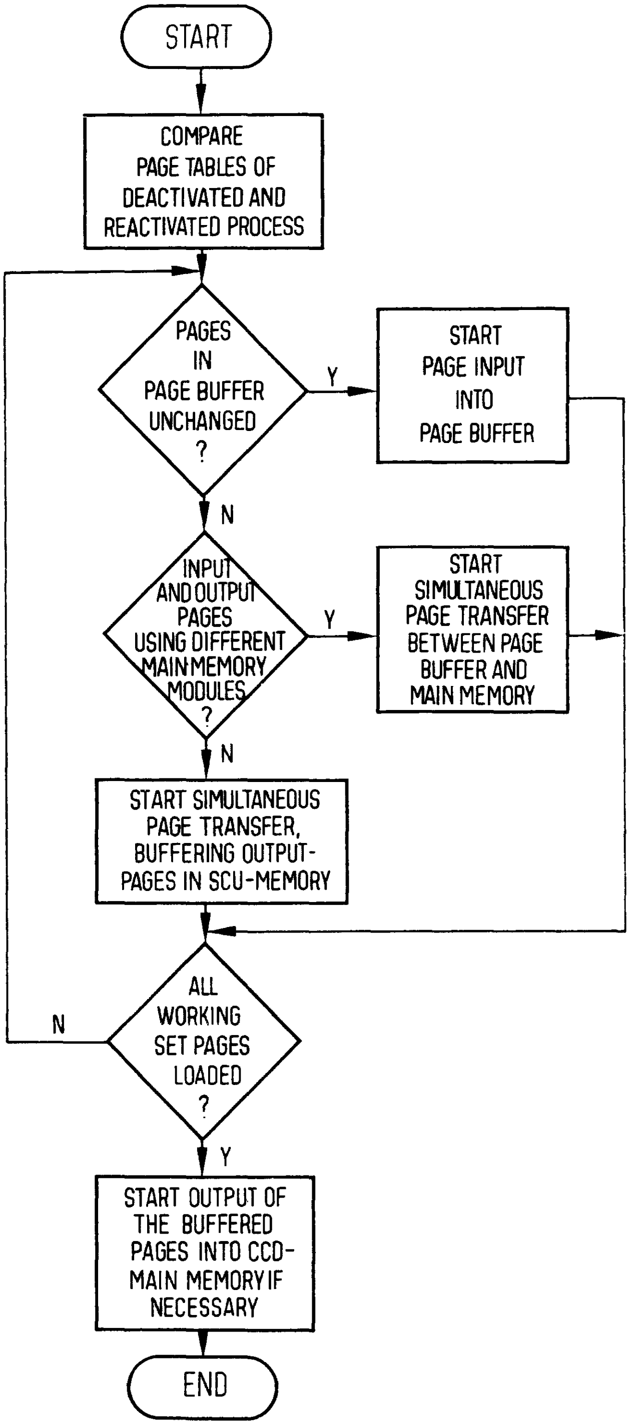

START
COMPARE PAGE TABLES OF DEACTIVATED AND REACTIVATED PROCESS
PAGES IN PAGE BUFFER UNCHANGED ?
Y
START PAGE INPUT INTO PAGE BUFFER
N
INPUT AND OUTPUT PAGES USING DIFFERENT MAINMEMORY MODULES ?
Y
START SIMULTANEOUS PAGE TRANSFER BETWEEN PAGE BUFFER AND MAIN MEMORY
N
START SIMULTANEOUS PAGE TRANSFER, BUFFERING OUTPUT-PAGES IN SCU-MEMORY
N
ALL WORKING SET PAGES LOADED ?
Y
START OUTPUT OF THE BUFFERED PAGES INTO CCD-MAIN MEMORY IF NECESSARY
END

Figure 5—Flow diagram of the working set restoration activity

The page buffer and main memory addresses are then switched through by the storage processor to the memory control, which is instructed to handle page transfer. Transferred first are those pages that can be entered in the page buffer simply by overwriting unmodified pages.

Transferred next are pages which lie in other modules than the pages to be displaced. This is done in the following fashion: the memory locations are read out of the page buffer in the Read/Update/Write mode and transferred to the main memory, while data are simultaneously written in from another main memory module.

The last pages to be entered are the ones which lie in the same module as the pages to be displaced. Here too, the data are entered in the page buffer via Read/Update/Write, but the page to be displaced from the page buffer must first be transferred to the storage processor for intermediate buffering. Pages of this type are entered in main memory after working set restoration has been completed.

Concurrently with page loading, the main memory addresses of the respective pages are entered in the currently unused portion of the hit/miss logic at a location corresponding to their page buffer address such that, at process change-over time, not only will the pages be in the page buffer, but the pertaining address information will also be stored in the hit/miss logic.

After this discussion of the three-level storage hierarchy's structure and functions, let us now examine its capability as compared with other systems.

CAPABILITY

To measure the capability of the three-level memory structure, a simulation program was created which allows an examination of the page miss rate in the page buffer for the transfer-on-demand strategy as well as under working set restoration. The different handling of page fault interrupts as compared with other interrupts has been taken into account. When a page fault occurs, all pages already entered in main memory are retained, whereas with all other interrupt causes the number of pages found after reactivation will be less. This is where the size of the main memory comes in; it can be calculated as follows: Assume that 100 users are concurrently attached to a computer system. Five of these are to be in the active process queue at any given time. Based on measured values, a mean working set size of 100 KByte can be assumed, which adds up to a memory capacity of about 500 KByte for the processes in the active queue. The remaining 95 users share the rest of the main memory capacity. If main memory capacity is 4 MByte, each process will find an average of 40 KByte available upon its reactivation. If each process is to find a greater capacity available, the main memory capacity must be increased if the length of the process queue remains the same. A greater main memory capacity will then result in increased capability since pages have to be overwritten less frequently in main memory and some of the page faults for previously used pages can be prevented. This is taken into account by the simulation program.

The analysis covered address sequences of several different programs. Table I uses two typical address sequences (program A and program B) to demonstrate the impact of main memory size on the page faulting rate and the page miss rate in the page buffer for both load strategies.

These results yielded by simulation were used for calculating the effective access times of useful processor accesses. The term "useful access" is employed to define only the number of accesses used for actual program processing, whereas the time needed for accesses during operating system activity is viewed as waiting time and is thus prorated to the useful accesses. Using this effective access time, the systems illustrated in Figures 1, 2, and 3 were analyzed and compared. The effect of the cache was ignored because it is the same in all systems.

The following formulae yield the effective access times for useful accesses.

For a two-level system

$$T_{eff1}=h_{pf}\cdot t_{int}+h_{int}\cdot t_{int}+(1\text{-}h_{pf}\text{-}h_{int})\cdot t_2, \text{ and}$$

for a three-level system

$$T_{eff2}=h_{pf}\cdot t_{int}+h_{int}\cdot t_{int}+(1\text{-}h_{pf}\text{-}h_{int}\text{-}m_2)\cdot t_2+m_2\cdot t_3$$

The same formula applies for the three-level systems which differ only in the load strategy. Depending on the load strategy, different miss probabilities (m_2') or (m_2'') must be expected in the page buffer.

h_{pf} and h_{int} represent the page faulting and interrupt rates, respectively; m_2 is the miss probability in the page buffer and t_{int} (=1 ms) represents the operating system execution time for process changeover. t_2 (=1 μs) is the access time to the second level of the two-level system, and t_2 (=160 ns) and t_3 (=51.2 μs) are, respectively, the access times to the second and third levels of the three-level system. Figures 6a and 6b show the cost curves of the compared systems: cost

TABLE I—Effect of main memory capacity on page faulting rate and page miss rate

Program	Main memory capacity	Interrupt frequency h_{int}	Page fault rate h_{pf}	Page miss rate transfer on demand (m_2')	Page miss rate working set rest. (m_2'')
	MB	%	%	%	%
A	4	0.07	1.67	4.66	0.93
	8	0.07	1.27	4.51	1.13
	12	0.07	0.93	4.41	1.30
B	4	0.07	0.38	0.61	0.00
	8	0.07	0.24	0.60	0.00
	12	0.07	0.18	0.60	0.01

curve C_2 applies for the two three-level systems differing only in their load strategy, while C_1 represents the conventional two-level system. As is clearly demonstrated, the costs of the three-level main memory are substantially lower. The figures also plot the effective access time curve of "useful access" for all comparison cases as a function of main memory capacity.

The effective access time of a three-level system with working set restoration T^{WSR}_{eff2} is almost equal to that of the two-level system T_{eff1}, Figure 6a, or can, according to Figure 6b, even be lower if no page buffer misses occur and the page buffer's significantly shorter access time, as compared with a conventional main memory, can thus be made the most of. The effective access time of a system with transfer on demand T^{TOD}_{eff2}, on the other hand, is always less favorable.

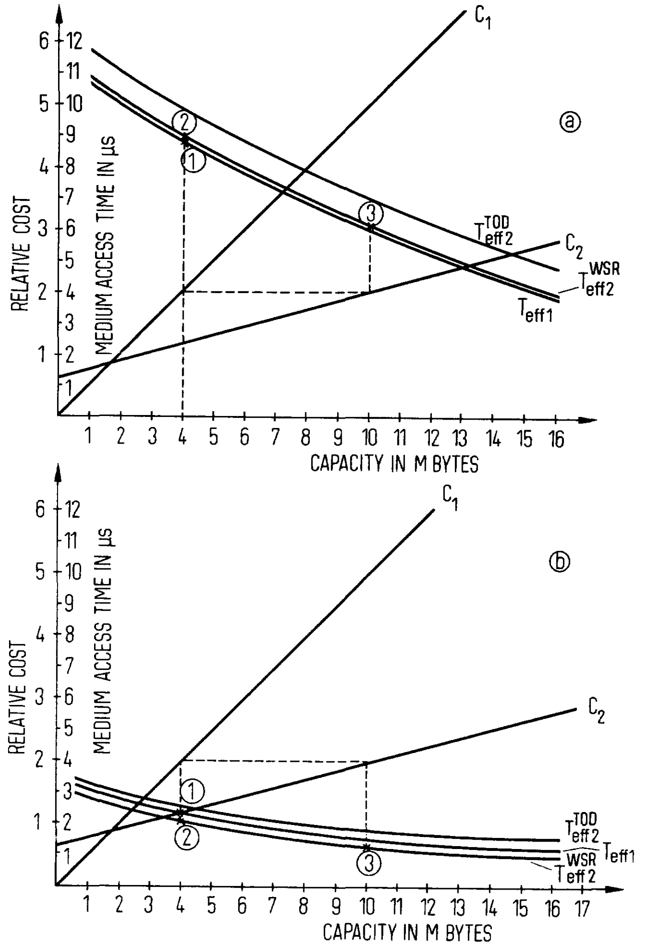


Figure 6—Cost and performance curves of the different memory systems

C_1: Cost of the two level hierarchy

C_2: Cost of the three level hierarchy with a 128 KByte page buffer

T_{eff1}: mean access time of the two level hierarchy

T^{WSR}_{eff2} and T^{TOD}_{eff2}: mean access times of the three level hierarchy under the different load strategies

(a) Simulation with program A

(b) Simulation with program B

CONCLUSION

Through the use of the working set restoration strategy, a three-level system can in fact always achieve nearly the same performance level as a conventional two-level system, provided that systems with equal main memory capacity—points 1 and 2—are juxtaposed. If, on the other hand, systems with equal cost levels are compared, the three-level system will always offer a more favorable effective access time (cf. points 1 and 3) due to the lower page faulting rate resulting from the greater main memory capacity.

ACKNOWLEDGMENT

The work has been supported under the Technological Program of the Federal Department of Research and Technology of the FRG. The author alone is responsible for the contents.

REFERENCES

1. Bhandakar, D. P., "Cost Performance Aspects of CCD Fast Auxiliary Memory," *Proc. CCD '75'*, Charge-Coupled Devices Appl. Conf., 1975, San Diego, pp. 435-442.
2. Boyle, W. S., and G. E. Smith, "Charge Coupled Semiconductor Devices," *Bell Syst. Tech. J.*, 49, 1970, pp. 587-593.
3. Ablassmeier, U. and E. Döring, *CCD—Schaltungen hoher Speicherdichte in Al-Si-Gate Technologie*, Siemens Forsch.- und Entwickl.-Ber. 4, 1975, pp. 226-230.
4. Schneider, P. and J. Witte, *CCD Memories in a Working Memory System*, Siemens Forsch.- und Entwickl.-Ber. 4, 1975, pp. 231-237.
5. Denning, P., "The Working Set Model for Program Behaviour," *Commun. of the ACM*, 11, 1968, pp. 323-333.
6. Tung, C., "On the Apparent Continuity of Processing in a Paging Environment," *IEEE Trans. Computers*, C-19, 1970, pp. 1047-1054.

Performance and power dissipation analysis for CCD memory systems

by S. L. REGE
Burroughs Corporation
Piscataway, New Jersey

ABSTRACT

In CCD memory systems a tradeoff exists between the frequency at which the memory system is operated and the power dissipation. The higher the frequency of operation, the lower is the service time and the higher is the power dissipation. A close look at the initial cost of the CCD memory system and the cost of maintaining these memory systems will show that the cost of maintenance for a year is nearly equal to the initial cost. This high cost necessitates an analysis of the CCD memory system design for service time and power dissipation.

In this analysis three different states called the Access, Refresh, and Idle states are defined for a CCD memory system. Each state is characterized by a frequency and five different modes of operations are defined depending on the relation between the frequencies. Average service time and power dissipation equations are then derived and each mode is analyzed. Contrary to the normal belief that the power dissipation increases with access frequency, it is shown that for certain modes the power dissipation is constant and is independent of frequency. Finally, a figure of merit is defined and the different modes are compared.

INTRODUCTION

In the last few years, extensive work has been done on Charge Coupled Devices (CCD's). These devices have a potential of becoming basic building blocks to construct memories for digital computers. Many papers are available that analyze and propose designs for the basic component and its layout on the chip.[1-3] Very little analysis is available on the use of the chip in a memory system. It is predicted that the cost/bit for CCD memory systems will be 15 to 30m¢/bit[4] and power dissipation of 20 uw/bit. Using a rule of thumb of 1 m¢/uw/year for the operation of semiconductor memory systems stated by Morton,[5] it is evident that the initial cost and the operating cost for a year are equal for CCD memory systems even when operated at low frequencies. The situation would be still worse at higher frequencies.

Here we will analyze some aspects of the CCD's pertaining to their use in a memory system design. The analysis will show the tradeoffs of operating and using different devices and an insight into the design of the chip.

CCD CHIP PARAMETERS

Most common types of CCD chips being designed are of the closed loop shift register type. Therefore, we will analyze such a chip. A typical chip will have S shift registers with N_t cells in each shift register (Figure 1). The design of the device will determine the refresh time (t_r), which is the maximum allowable time before which a new refresh cycle must be started. Usually a refresh amplifier is available after every N_r cells, and N_t is an integer multiple of N_r, with N_r and

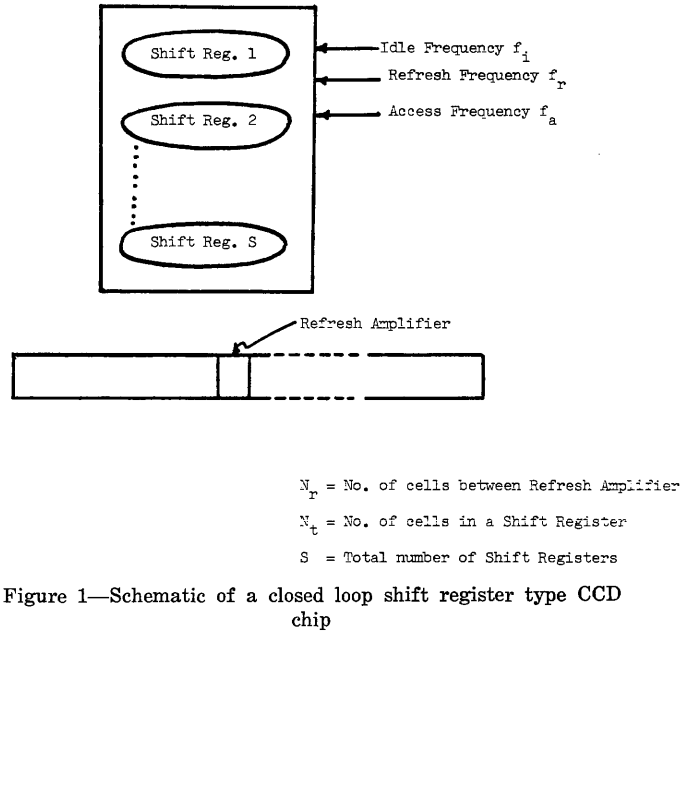

Shift Reg. 1
Shift Reg. 2
Shift Reg. S
Idle Frequency f_i
Refresh Frequency f_r
Access Frequency f_a
Refresh Amplifier
N_r = No. of cells between Refresh Amplifier
N_t = No. of cells in a Shift Register
S = Total number of Shift Registers

Figure 1—Schematic of a closed loop shift register type CCD chip

Chip Parameter	Symbol	Typical Values
Shift Registers	S	32∼128
No. of Cells/Shift Register	N_t	64∼256
No. of Cells between Refresh Amp.	N_r	32∼256
Refresh Time	t_r	2msec. and above

TABLE I—Various Chip Parameters and Their Typical Values

N_t being powers of two. Some typical values for S, N_t, N_r and t_r are given in Table I.

MODES OF OPERATION

With the above mentioned basic parameters for the chip a memory system can now be designed to operate in three different states. The three states are the Access state when the data is being accessed (read or written) from the chip, Idle state in which the chip is doing nothing and the Refresh state in which the data in the shift register is being refreshed.* The three states can be characterized by having three different frequencies: Access frequency f_a, Idle frequency f_i and Refresh frequency f_r. By definition, a memory system can be in the Access state and Refresh state at the same time if the frequencies for these two states are the same. A state transition diagram for the three states is shown in Figure 2.

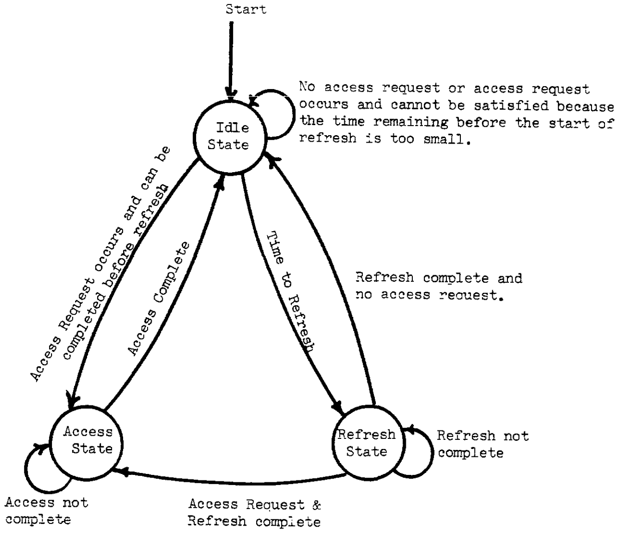

Start
Idle State
No access request or access request occurs and cannot be satisfied because the time remaining before the start of refresh is too small.
Access Request occurs and can be completed before refresh
Access Complete
Time to Refresh
Refresh complete and no access request.
Access State
Refresh State
Refresh not complete
Access not complete
Access Request & Refresh complete

Figure 2—State and transition diagram for the different states of CCD memory operation

In evaluating the performance of the chip, the following assumptions will be made:

(1) The refresh state has priority over both the access and idle state, whereas the access state has priority over the idle state.

(2) The data transfer in a shift register is always initiated from the bit addressed as the first bit. This is not always necessary and different assumptions can be treated as special cases and the analysis pursued here modified accordingly.

(3) A new access for data is not made when a previous access is pending. This assumption implies a buffer in which all the requests to the memory system are stored and serviced with some scheduling strategy.

(4) There is an equal probability that the refresh, idle or access states are started at any bit in the shift register.

(5) The power dissipation is proportional to the frequency at which the data cells are being moved.[6] Notice that the power dissipation of the peripheral circuitry, drivers and the shift register array is a function of the frequency.[1]

(6) The refreshing and accessing can be done simultaneously, if, and only if, the refreshing frequency (f_r) and the accessing frequency (f_a) are the same. This implies that the interface to the CCD memory system always transfers or receives data at the same frequency.

(7) Once a data transfer is started all the bits in a shift register must be transferred.

By choosing different values for the three frequencies f_a, f_r and f_i, the chip and hence, the memory system can be operated in five different modes (Table II) requiring one, two, or three clocks. Notice that the number of clocks required is an indication of the complexity of control required by the memory system and, therefore, in some sense defines the cost for control circuitry.

The five modes of operation and their implications are tabulated in Table II. Mode 5 is the most general mode of operation and the other modes can be considered as a limiting case of this mode. For example, mode 3 can be considered as a limiting case of mode 5, when f_i tends to f_r.

Before evaluating the different modes, the implications of the modes of operation on a memory system design are qualitatively discussed. In a semiconductor memory system design some major parameters of interest are: (a) power dissipation which influences the cost of maintaining the memory, (b) access and service times which influence the performance of the system, (c) the interface requirements such as, the data width and data rate, and (d) the control complex-

* The Access state may be further divided into an Aligning state and a Transfer state. Presently, we will not make any distinction and the analysis made here can be easily extended to that case.

MODE	FREQUENCY	METHOD OF OPERATION
1	$f_a = f_r = f_i$	Continuous Refresh Most common mode of operation in the near future.
2	$f_a \neq f_r$ $f_a = f_i$	Burst Refresh Possible low access frequency due to interface requirements.
3	$f_a \neq f_r$ $f_i = f_r$	Possible high Access rate. Useful when the memory is expected to remain idle for a large percentage of the time.
4	$f_a \neq f_i$ $f_a = f_r$	Zero idle frequency is possible. Minimum access frequency determined by the lowest allowable refresh frequency.
5	$f_a \neq f_i \neq f_r$	Zero idle frequency possible. Interface requirement of low access frequency can be satisfied.

TABLE II—Possible Modes of Operations for a CCD Chip

ity which determines the difficulty and the amount of overhead involved in the design of the memory system.

Mode 1 is a continuous mode of operation in which all the bits are moving all the time and has the simplest control circuitry. This will be the most common mode of operation in the early systems.

Mode 2 requires two frequencies and, therefore, has a control complexity higher than Mode 1. One method of operating in Mode 2 would be to use low idle and access frequencies and high refresh frequency. Such a mode is applicable when an external device with low data rate is interfaced with CCD's and real time transfers are made. Also, when memory is operated in a 'vertical mode',[7] as is recommended for CCD device,[8] it is necessary to shift the bits at a low frequency.

Mode 3 of operation has equal idle and refresh frequency and either low or high access rate. The control complexity is the same as in Mode 2. It is necessary that $f_r(=f_i) > N_r/t_r$. Intuitively, such a design is attractive only when the memory system is expected to remain in the idle mode for a high percentage of the time. Mode 4 of operation is again as complex as Mode 2 and Mode 3 and can be operated with burst refresh and zero idle frequency. Finally, Mode 5 is the most general mode of operation and will have the maximum control complexity. A practical application of this would be: (1) when the system is idle most of the time, (2) the external requirement necessitates low data rate, and (3) burst refresh is to be used to refresh the memory. The analysis later will show that only some of these modes are advantageous from the performance and power dissipation considerations.

On the following pages, equations for Mode 5 of operation will be developed. Then the equations for the other modes of operation will be determined as a limiting case of Mode 5.

ANALYSIS OF SERVICE TIME

Service time is defined as the time elapsed between the moment a request is made to the memory for some information to the moment when all the information is delivered.

When a request is made to a CCD memory system for particular information, the bits in the shift registers have to be shifted until the shift register is positioned at bit 1 under the read/write circuit. The time required to do this will be called the access time (t_a). Once the shift register has been positioned then the data may be transferred and the time required to transfer the data will be called the transfer time (t_t). The service time (t_s) is the time required to service a request and is given by

$$t_s = t_a + t_t$$

and the average value of service time ($\bar{t}_s$) is

$$\bar{t}_s = \bar{t}_a + \bar{t}_t$$

Assume that t_a is a random variable and can be determined as a sum of three random variables given below.

X_r = random variable that represents time spent in refreshing when a service request is made.

X_a = random variable that represents time spent in aligning the shift register to bit 1 when a service request occurs.

X_d = random variable that represents time lost when a service request cannot be satisfied because the refresh cycle has to be started before all the bits in the shift register are transferred.

Then

$$t_a = X_r + X_a + X_d$$

and because X_r, X_a and X_d are independent random variables, we have

$$\bar{t}_a = \bar{X}_r + \bar{X}_a + \bar{X}_d$$

We have to determine the values of X_r, X_a and X_d. The probability density function of X_r is given in Figure 3. From the figure

$$\bar{X}_r = \frac{1}{2}\left(\frac{N_r}{f_r}\right)^2 \cdot \frac{1}{t_r}$$

The value for X_a is simple to calculate and is given by

$$\bar{X}_a = \frac{1}{2}\frac{(N_t-1)}{f_a} \approx \tfrac{1}{2}(N_t/f_a) \qquad \text{[for large } N_t\text{]}$$

To calculate $\bar{X}_d$, notice that if a request arrives at a time when the shift register is at position N, then the minimum time required to access and transfer the block is given by

$$\frac{N_t + (N_t - X)}{f_a} = \frac{2N_t - X}{f_a}$$

Because the probability that a request occurs at any particular bit is the same, the expected value of X_d is (Figure 4) derived as

$$\bar{X}_d = \sum_{X=1}^{N_t} \frac{1}{N_t t_r}\left[\frac{1}{2}\,\frac{(2N_t - X)^2}{f_a^2} + \frac{N_r(2N_t - X)}{f_r f_a}\right]$$

$$= \frac{1}{2t_r f_a^2}\left[\frac{7}{3}N_t^2 - \frac{3}{2}N_t + \frac{1}{6}\right] + \frac{N_r}{N_t t_r f_r f_a}\left[\frac{3N_t^2 - 2N_t}{2}\right]$$

Notice that for large values of N_t

$$\left(\frac{3}{2}N_t - \frac{1}{6}\right) \ll \frac{7}{3}N_t^2 \text{ and } 2N_t \ll 3N_t^2$$

$$\bar{X}_d \approx \frac{7}{6t_r}\left(\frac{N_t}{f_a}\right)^2 + \frac{3N_t N_r}{2t_r f_r f_a}$$

Then the average value of t_a is

$$\bar{t}_a = \frac{N_t}{2f_a} + \frac{1}{2t_r}\left[\left(\frac{N_r}{f_r}\right)^2 + \frac{7}{3}\left(\frac{N_t}{f_a}\right)^2 + \frac{3N_t N_r}{f_a f_r}\right]$$

For a shift register of length N_t and transfer frequency f_a

$$\bar{t}_t = \frac{N_t}{f_a}$$

Therefore, the average value of the service time is

$$\bar{t}_s = \frac{3N_t}{2f_a} + \frac{1}{2t_r}\left[\left(\frac{N_r}{f_r}\right)^2 + \frac{7}{3}\left(\frac{N_t}{f_a}\right)^2 + \frac{3N_t N_r}{f_a f_r}\right]$$

ANALYSIS OF POWER DISSIPATION

In CCD's power dissipation is proportional to the frequency.[6] Amelio[1] gives equation for the average power dissipated (P_d) on chip for a data frequency f as

$$P_d = 2N(0.2 \times 10^{-12})f$$

where P_d is the power dissipated due to a shift register of N bits being shifted at a frequency f.

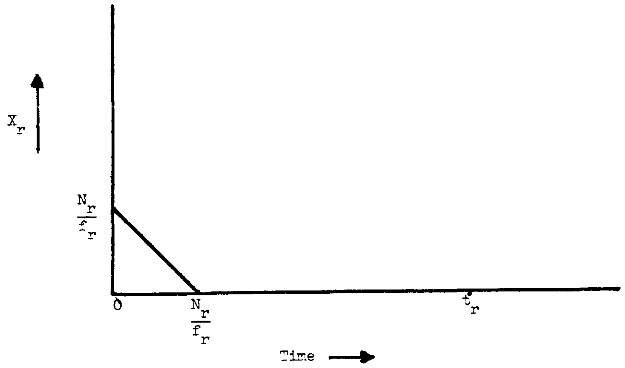


Figure 3—Probability density function of X_r

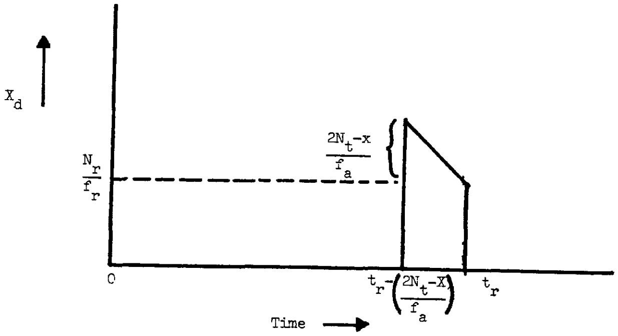


Figure 4—Probability density function for X_d when a service request is made at bit position x

Because the CCD electrodes have a large capacitive load, the power dissipated in the drivers is much larger and is given by

$$P_d = CV^2f$$

where C is the capacitive load, and V is the voltage applied.

We will represent the constant of proportionality between power dissipation and frequency as K. Usually the power dissipation due to the drivers is much larger than the power dissipation due to the shift register array and hence, $K=CV^2$.

Then if $t_1, t_2, t_3 \ldots t_n$ are different time intervals in which frequencies $f_1, f_2, \ldots f_n$ are applied to the chip, the average power dissipation is given by

$$\bar{P}_d = K\frac{t_1f_1 + t_2f_2 + \ldots\ldots\ldots + t_nf_n}{t_1 + t_2 + \ldots\ldots\ldots + t_n}$$

The three different states of the CCD memory systems will determine three different time intervals and three different frequencies.

These are:

(1) Total time spent in refresh state (T_r) and refresh frequency f_r.

(2) Total time spent in idle state (T_i) and idle frequency f_i.

(3) Total time spent in access state (T_a) and access frequency f_a.

Then the average power dissipation ($\bar{P}_d$) is given by

$$\bar{P}_d = K\frac{f_r\bar{T}_r + f_i\bar{T}_i + f_a\bar{T}_a}{\bar{T}_r + \bar{T}_i + \bar{T}_a}$$

To determine the average values for the different times we will consider a basic period as the time from the beginning of one Refresh cycle to the beginning of the next Refresh cycle. Then

$$\bar{T}_r = \frac{N_r}{f_r}$$

The remaining time in a cycle is the sum of the total idle time and the total service time. Therefore

$$T_a + T_i = \left(t_r - \frac{N_r}{f_r}\right)$$

$$\bar{T}_i = \left(t_r - \frac{N_r}{f_r}\right) - \bar{T}_a$$

To determine the average total time spent in servicing requests ($\bar{T}_a$) assume that in any given cycle there is a possibility of maximum of s accesses. Then let

p(j) = probability that j accesses are made

(j = 0, 1, 2 . . . s)

The average time spent when one access is made is already determined and is given by $\frac{3N_t}{2f_a}$ and, therefore,

$$T_a = \sum_{j=0}^{s} j\,p(j)\frac{3N_t}{f_a}$$

$$\text{Let } Q = \sum_{j=0}^{s} j\,p(j)$$

Intuitively Q represents the average number of requests per period.

Then

$$\bar{T}_a = \frac{3N_tQ}{2f_a}$$

and

$$\bar{T}_i = t_r - \frac{N_r}{f_r} - \frac{3N_t}{2f_a}Q$$

Adding the various terms and rearranging gives the average power dissipation for mode 5 of operation as

$$\bar{P}_d = K\left\{f_i + \frac{N_r}{t_r}\left(1 - \frac{f_i}{f_r}\right) + \frac{3N_tQ}{2t_r}\left(1 - \frac{f_i}{f_a}\right)\right\}$$

This general analysis for average service time and power dissipation can now be applied to each mode of operation.

ANALYSIS OF DIFFERENT MODES

In this section we will analyze each mode of operation. The service time and power dissipation equations for the different modes derived from the analysis of general Mode 5 are given in Table III. The graph in Figure 5 denotes the average service time and power dissipation for Mode 1 of operation and the graphs in Figures 6 to 10 denote the percentage improvement in average power dissipation and percentage degradation in service time for Modes 2 to 5 over that of Mode 1 of operation.

Mode 1 will be the most usual mode of operation in the near future. A plot of access frequency vs service

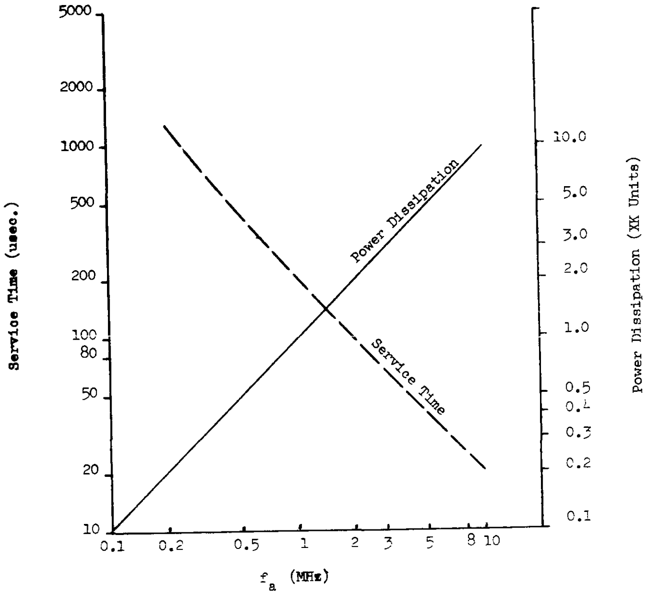


Figure 5—Graph of service time and power dissipation vs. access frequency for mode 1 of operation

MODE	FREQUENCY	SERVICE TIME	POWER DISSIPATION	REMARKS
1	$f_a=f_r=f_i$	$\frac{3N_t}{2f_a}+\frac{1}{2t_r}\left\{\frac{3N_r^2+7N_t^2+9N_tN_r}{3f_a^2}\right\}$	Kf_a	Power Dissipation directly proportional to frequency
2	$f_a\neq f_r$ $f_a=f_i$	$\frac{3N_t}{2f_a}+\frac{1}{2t_r}\left\{\frac{N_r^2}{f_r^2}+\left[\frac{7}{3}\ \frac{N_t^2}{f_a^2}\right]+\frac{3N_tN_r}{f_af_r}\right\}$	$K\left\{f_a+\frac{N_r}{t_r}\left[1-\frac{f_a}{f_r}\right]\right\}$	Very little advantage over Mode 1 for both service time or power dissipation
3	$f_a\neq f_r$ $f_r=f_i$	Same as Mode 2	$K\left\{f_r+\frac{3N_tQ}{2t_r}\left[1-\frac{f_r}{f_a}\right]\right\}$	Values of f_i=1 to 3Mcycles/sec. are advantageous from both service time and power dissipation considerations
4	$f_a\neq f_i$ $f_a=f_r$	Same as Mode 1	$K\left\{\left[1-\frac{2N_r+3N_tQ}{2f_at_r}\right]f_i+\frac{2N_r+3N_tQ}{2t_r}\right\}$	For f_i = 0 and a given value of Q, the power dissipation is constant
5	$f_a\neq f_r\neq f_i$	Same as Mode 2	$K\left\{f_i+\frac{N_r}{t_r}\left[1-\frac{f_i}{f_r}\right]+\frac{3N_tQ}{2t_r}\left[1-\frac{f_i}{f_a}\right]\right\}$	For f_i=0 and a given value of Q, the power dissipation is constant. No advantage over Mode 4 unless $f_a\neq f_r$ due to interface requirements

TABLE III—Service Time and Power Dissipation Equations for Different Modes of Operation

time is given in Figure 5. The dominating factor for service time in the equation given in Table III is $3N_t/2f_a$. At small values of access frequency (f_a=200KHz) the contribution due to the second term is quite high (about 40 percent), whereas at moderate values (f_a=1MHz) it is about 10 percent and at high values it is negligible (less than 1 percent). The second term can be decreased by increasing t_r. N_r has no appreciable influence on the service time. The power dissipation is seen to be directly proportional to frequency.

Again, for Mode 2, the dominant term for service time is $3N_t/2f_a$. The power dissipation equation shown in Table III has two terms. The first term is the same as Mode 1 and the second term can be made negative by making $f_a>f_r$.

Figure 6 shows the percentage improvement in power dissipation and percentage degradation in service time over Mode 1 vs the access frequency for different values of f_r. The results show that either the degradation in service time is quite high or power dissipation improvement is quite low. A similar analysis for various values of N_r will show the same conclusion. Therefore, this mode of operation has little advantage over Mode 1. The service time equation for Mode 3 is the same as that for Mode 2 but the power dissipation equation is different.

Figure 7 shows the percentage improvement in power dissipation and percentage degradation in service time vs the access frequency for various values of Q. Service time is independent of Q but the improvement in power dissipation is reduced as Q increases. The difference in $\overline{P}_d$ improvement between the lowest and the highest value of Q is quite small and is of the order of 10 to 15 percent. Note that there is a rapid improvement in power dissipation from 1 to 5MHz and then the improvement tapers off. The worst case service time degradation is about 8 percent. Therefore, a good cut-off point for this mode of operation is around 5MHz when the service time degradation is about 5 percent.

The variation of percentage improvement in power dissipation and degradation in service time vs the access frequency for various values of f_r is shown in Figure 8. It shows that high values of f_r are disadvantageous from both power dissipation and service time point of view. Generally, this mode is better than Mode 2 of operation.

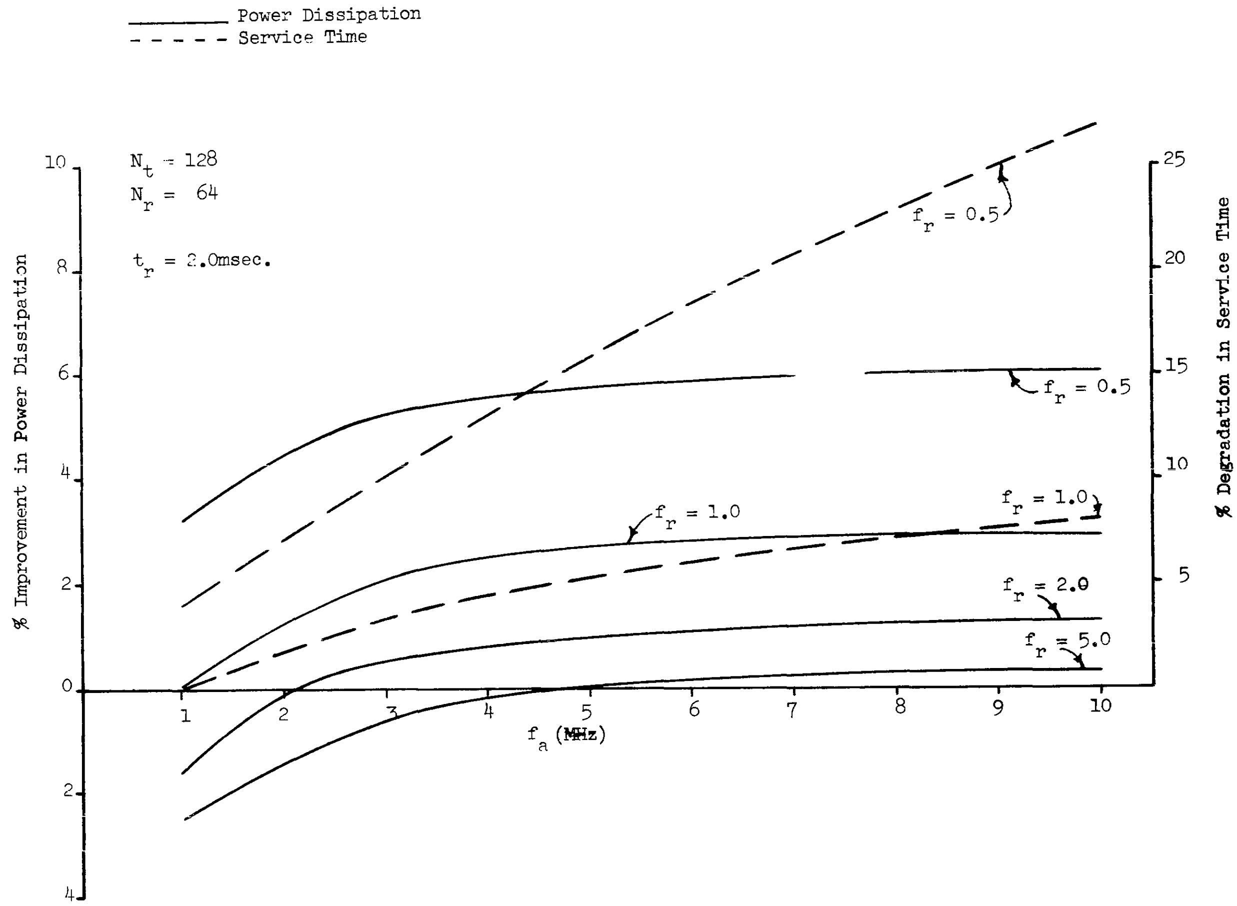


Figure 6—Percentage improvement in power dissipation and percentage degradation in service time vs access frequency for mode 2 of operation

In the Mode 4 of operation the average service time equation is the same as Mode 1. Physically, this is understandable due to two reasons: the idle frequency moves the information bits without doing any useful work and there is an equal probability of a service request at any bit.

Power dissipation equation has two terms, one proportional to the frequency f_i and the other a constant. By making f_i zero, and if the different parameters of a chip (e.g. N_a, t_r, etc.) are given, then power dissipation is only dependent on Q and is independent of the access frequency. Notice that increasing t_r reduces the power dissipation. This is again physically understandable, since the chip will have to be refreshed less regularly. The power dissipation also can be decreased by decreasing N_r or N_t. Notice that the second term can never be negative. Therefore, the minimum value of power dissipation occurs when $f_i=0$. Because the most interesting point is $f_i=0$, we will further analyze this mode at this operating point. With $f_i=0$ the power dissipation for Mode 4 and Mode 5 are the same.

Figure 9 shows the percentage improvement in power dissipation and percentage degradation in service time vs the access frequency for various values of Q for Modes 4 and 5. The percentage improvement in power dissipation increases as access frequency increases. Increase in the value of Q decreases the percentage power dissipation. For smaller values of access frequency ($f_a=1$ to 3MHz) the slope of the power dissipation curve is quite high (40) whereas at higher values it is small (2). The power dissipation improvement for values of f_a greater than 3.5 to 4MHz is marginal. Therefore, an optimal point of operation is $f_a=3$MHz for small values of Q and $f_a=4.5$ to 5MHz for large values of Q.

Finally, the following observations can be made:

(1) It is possible to operate a memory system at the highest possible frequency with about 90 percent improvement in power dissipation over Mode 1 of operation.

(2) It is advisable to make the time between refreshing (t_r) as large as possible both to decrease the average power dissipation and service time.

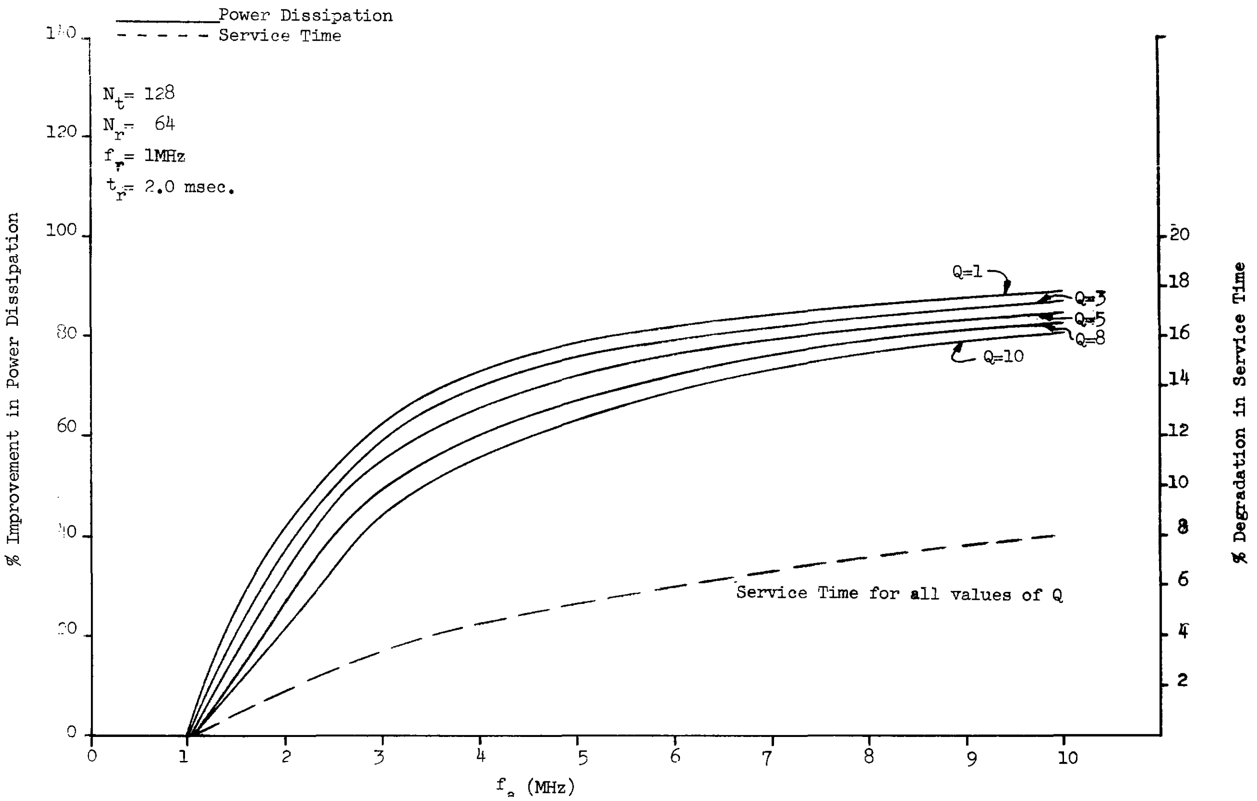


Figure 7—Percentage improvement in power dissipation and percentage degradation in service time for various values of Q for mode 3 of operation

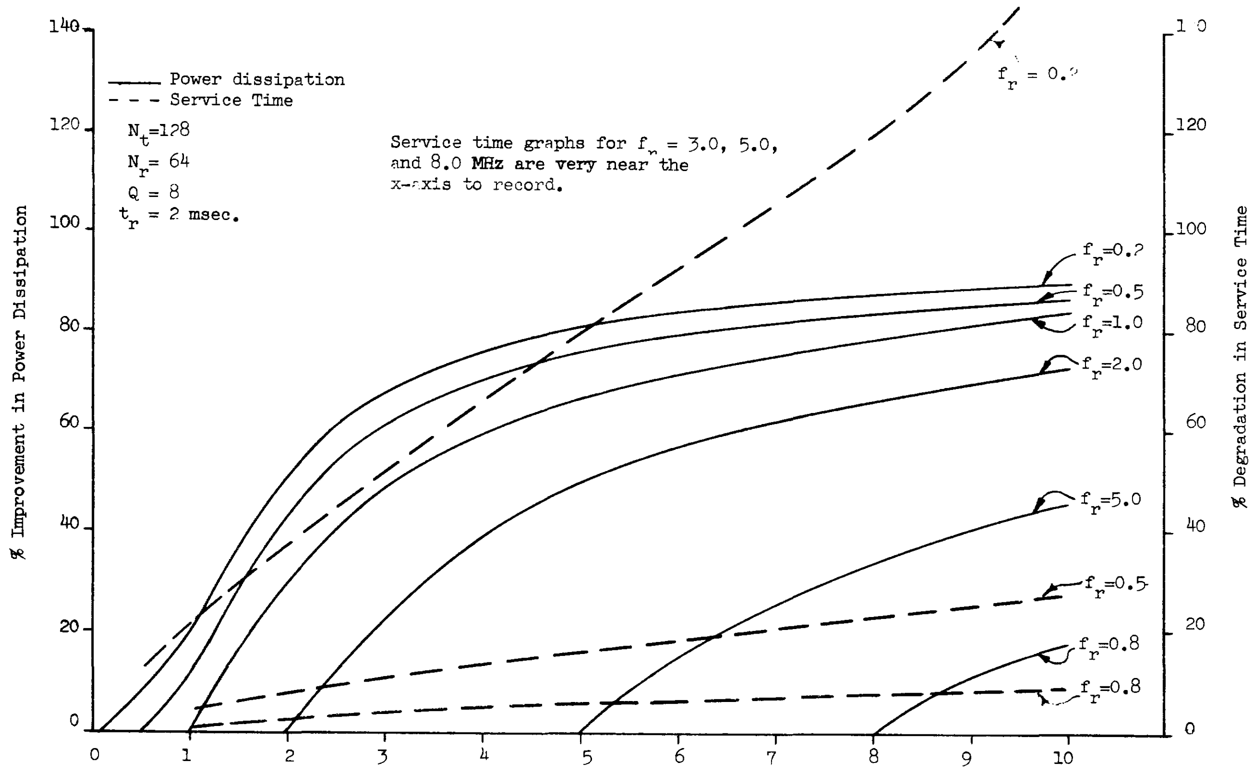
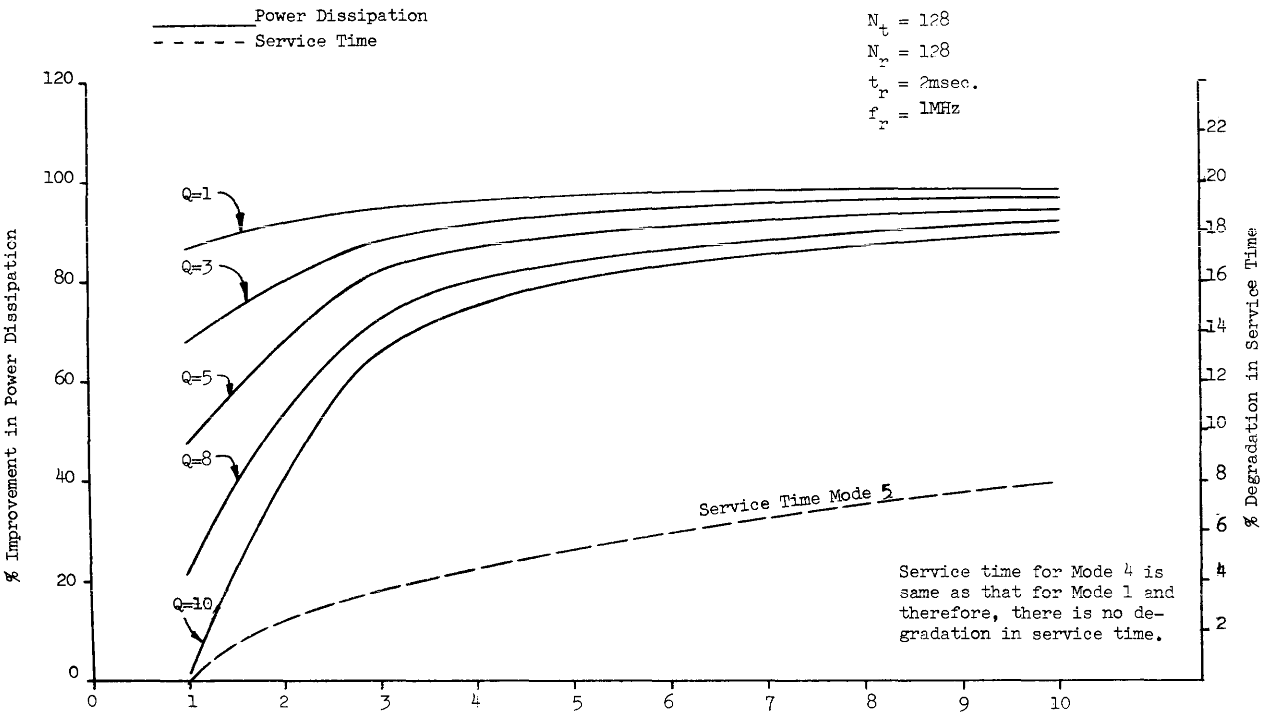


Figure 8—Percentage improvement in power dissipation and percentage degradation in service time vs access frequency for various values of f_r for mode 3 of operation



Service time for Mode 4 is same as that for Mode 1 and therefore, there is no degradation in service time.

Figure 9—Percentage improvement in power dissipation and percentage degradation in service time for mode 4 and mode 5 of operation

(3) The interesting modes of operation are derived by making idle frequency zero. This is intuitively valid because moving the bits in a shift register during idling does not have any advantage either from performance or from power dissipation viewpoint.

COMPARISON OF DIFFERENT MODES

In this section we will compare the different modes of operation. To make a comparison we will define a figure of merit (Fgm) as the number of possible services per unit time per unit power dissipation.

$$\text{Fgm} = \frac{1}{\text{average service time} \times \text{average power dissipation}}$$

Thus, a mode that has higher Fgm is better than one that has a lower Fgm.

In Figure 10 we draw a graph of Fgm vs. Access frequency for various modes. The typical values chosen for N_t, N_r, and t_r are 128, 64 and 2msec. The Refresh frequency, whenever it is different from Access frequency is chosen as 1MHz. The Mode 1 and Mode 2 are independent of Q, whereas Modes 3, 4, and 5 are dependent on Q and, therefore, graphs are drawn for various values of Q.

Mode 2 has the worst Fgm for all access frequencies and, therefore, is the worst mode of operation. Mode 1 has a typical Fgm of about 5 and is constant over all frequencies. For values of $f_a > 1$MHz Modes 3 and 4 and Mode 5 are better than Mode 1 for all values of Q up to 10. For $f_a = 1$MHz and values of Q=1,5, Mode 1 of operation is better than Mode 3. Also, for Q=10 Mode 1 is better than Mode 4 or Mode 5. But notice that for Q=10 Mode 3 is the best mode of operation for small values of f_a. At higher values of f_a Mode 4 is the best mode, closely followed by Mode 5 for all values of Q. The Fgm for Modes 3, 4 and 5 is reduced as Q increases. The reduction for Modes 4 and 5 is much higher than that for Mode 3.

Table IV lists the various modes and a qualitative comparison of these modes with respect to control complexity and cost of design.

CONCLUSIONS

Memory systems built with CCD's are shown to have three states of operation: The Access state, Refresh state and Idle state and each state has a frequency associated with it. Three different modes of operation are defined and average service time and average

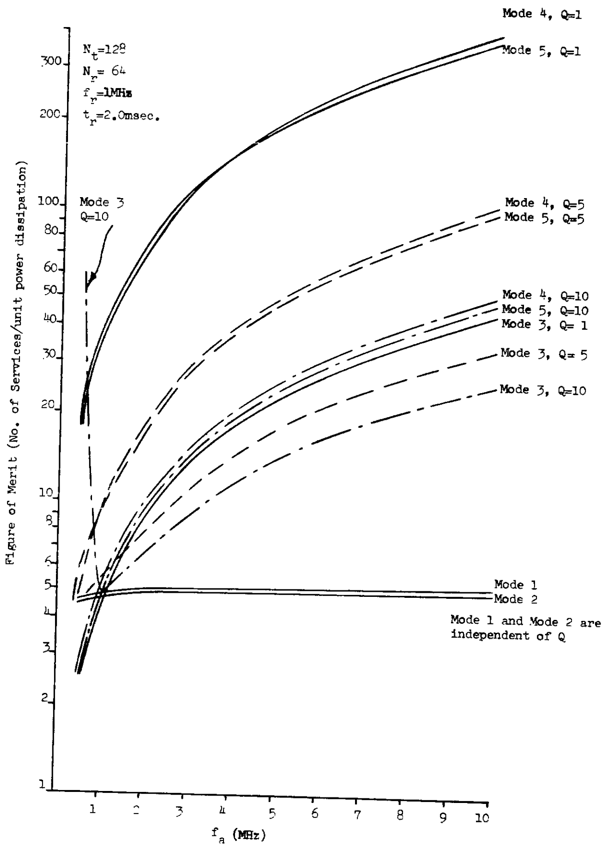


Figure 10—Figure of merit vs access frequency for various modes of operation

TABLE IV—A Qualitative Comparison of Different Modes of Operation

MODE	NO. OF CLOCKS REQUIRED	CONTROL COMPLEXITY	COST OF DESIGN	REMARKS
1	1	Minimum	Minimum	Simplest mode of operation
2	2	Moderately Low	Moderate Lower than Mode 4 if $f_i \neq 0$	Worst mode of operation
3	2	Moderately High	Moderate	For small f_a (f_a <1MHz) . A good mode of operation
4	2 if $f_i \neq 0$ 1 if $f_i = 0$	High if $f_i \neq 0$ Low if $f_i = 0$	Low if $f_i = 0$	For f_a > 1MHz the best mode of operation
5	3 if $f_i \neq 0$ 2 if $f_i = 0$	Maximum	Maximum	Best mode of operation except for mode 4

power dissipation equations for a general case of operation are derived. Equations for each mode are then derived as a limiting case of these general equations. The different modes are analyzed individually and, finally, a comparison between the different modes is made by defining a figure of merit. An interesting result derived is that the power dissipation is constant and is independent of Access frequency for Modes 3 and 4 when they are operated with $f_i=0$. Mode 4 is shown to be the best mode of operation. A simple qualitative comparison is finally made for the cost of implementation.

ACKNOWLEDGMENT

I would like to thank Dr. Bob Woo for extensive discussions during the development of this work, Mr. Paul White for comments that improved the quality of the paper, and Marie Golden for typing this paper.

REFERENCES

1. Amelio, G. F., "Charge Coupled Devices for Memory Applications," *AFIPS Conference Proceedings*, Vol. 44, May 1975, pp. 515-522.
2. Carnes, J. E. and W. F. Kosonocky, "Charge Coupled Devices for Computer Memories," *AFIPS Conference Proceedings*, Vol. 43, May 1974, p. 833.
3. Collins, D. R., J. B. Barton, D. C. Buss, A. R. Kinetz, S. E. Schroeder, "CCD Memory Options," *1973 IEEE Solid-State Circuits Conference*, p. 136.
4. Martin, R. R. and H. D. Frankel, "Electronic Disks in the 1980's," *Computer*, February 1975, p. 28.
5. Morton, J. A., "Strategy for Memory System Technology," *IEEE Transactions on Magnetics*, September 1971, pp. 333-336.
6. French, B. T., "Designers Guide to Charge Coupled Devices," *EDN*, Jan. 20, 1972, pp. 34-38.
7. R. Papenberg, *Applications Using Intel's 2416 16K Charge Coupled Device*, Application Notes, Intel Corporation, 3065 Bowers Avenue, Santa Clara, Calif. 95051.
8. Intel 16,384 Bit CCD Serial Memory, Specification Sheet, Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051.

Intelligent memory

by MURRAY EDELBERG and L. ROBERT SCHISSLER
Sperry Research Center
Sudbury, Massachusetts

ABSTRACT

The intelligent memory is a computer memory formed of circulating serial storage loops and distributed processing logic. In addition to the basic information storage function, the memory performs off-line sort processing, associative searching, updating and retrieval. The memory is also capable of dynamically varying its loop size to accommodate varying data requirements.

A number of memory configurations which trade performance for economy are possible. The options range from single record per loop and on-chip logic (aimed at CCD technology) to multiple records per loop and off-chip logic (aimed at magnetic bubble memories). The latter option is made possible by a new sort algorithm named "gyro sort" in which loop contents are caused to "precess" at appropriate intervals.

As one component of a storage hierarchy, the intelligent memory offers potential performance gains ranging from one to three orders of magnitude over random access memories at comparable cost.

STRUCTURE

The memory contains one or more identical modules. Figure 1 gives a block diagram of such a module, including $n \geq 2$ storage loops labeled $L_1, L_2, \ldots, L_n$ and n processing elements labeled $PE_1, PE_2, \ldots, PE_n$, where n is even. Each storage loop is a circulating shift register. All storage loops are shifted synchronously using a common clock. Processing element PE_i is incident with storage loops L_i and L_{i+1}.

There are two loop connection states possible within a processing element. Referring to Figure 2, the loop connection state indicated by the solid lines within processing element PE_1 is the *thru* state and isolates storage loop L_1 and L_2. The dashed lines indicate the *interchange* state which, if held for one complete rotation of loops L_1 and L_2, effects an interchange of the contents of these loops. The interchange connection, if held indefinitely, effectively joins the two incident loops into a single loop of double length. By causing $k-1$ contiguous processing elements to hold the interchange connection, a storage loop of size k times the basic loop size is formed.

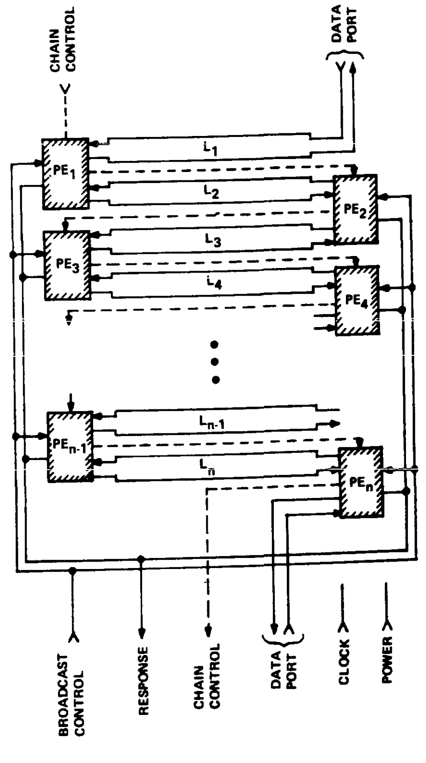


Figure 1—Intelligent memory module block diagram

A processing element is *inactive* if it is simply maintaining the interchange connection indefinitely and *active* otherwise. The pattern of active (denoted A) and inactive (denoted I) PEs is restricted to be of the form:

$$\underbrace{\underbrace{\text{II}\ldots\text{I}}_{k-1}\ \text{A}\ \underbrace{\text{II}\ldots\text{I}}_{k-1}\ \text{A}\ \ldots\ \underbrace{\text{II}\ldots\text{I}}_{k-1}\ \text{A}}_{n/k}$$

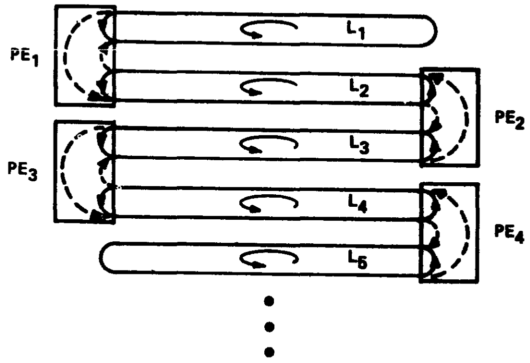


Figure 2—Loop connection states

Active PEs fall into two phases: the odd phase consists of the 1st, 3rd, 5th, . . . active PEs and the even phase contains those remaining.

Figure 1 shows distribution of control signals to PEs within a module via chain control lines (shown dashed) and broadcast control lines. Broadcast control lines send the same information to all PEs; chain control lines send information which may be modified by PE_i before being passed on to PE_{i+1}. Activity status is assigned to PEs via chain control lines. Certain broadcast control signals are made operative only at active PEs. Moreover, broadcast control signals designated odd (even) are made operative only at active PEs belonging to the odd (even) phase. A response line ORs together responses from all active PEs in the module. When two or more modules are used in a memory system, identical broadcast control signals are presented to each, and response signals from each are OR'ed together. Modules are connected in series with respect to their chain control lines and data ports; special terminations at the extremes are required.

Table I lists broadcast control modes. Table II lists individual broadcast control, chain control and response signals at the module level.

Figure 3 shows a block diagram of the general processing element PE_i. Signals internal to PE_i are identified in Table III.

TABLE I—Broadcast Control Modes

SORT <
SORT >
SEARCH =
SEARCH >
SEARCH <
UPDATE
IDLE
RECONFIGURE

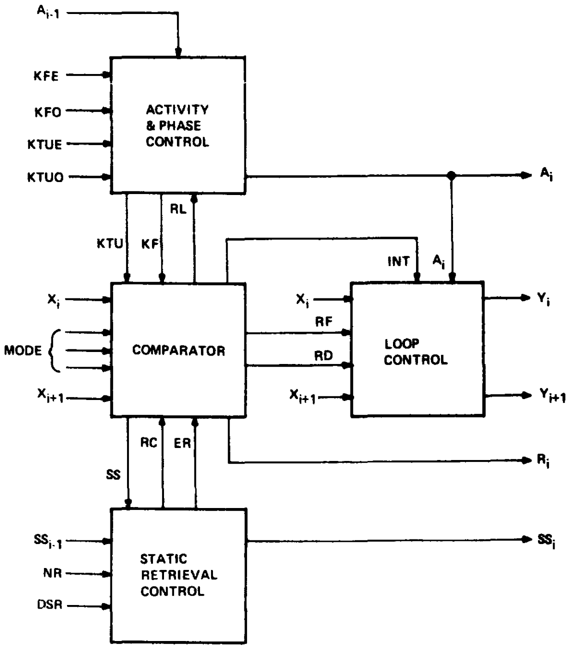


Figure 3—Processing element PE_i block diagram

In Figure 3, inputs from (outputs to) loops L_i and L_{i+1} are labeled x_i, x_{i+1} (y_i, y_{i+1}) respectively. Four main components of PE_i are shown: loop control (maintains appropriate loop connection state between loops L_i and L_{i+1}), activity and phase control (maintains activity status and computes phase for PE_i), static retrieval control (determines whether PE_i is a first responder during search operations), and the comparator (performs serial comparison and computes internal control signals).

We estimate PE complexity using MOS logic to be roughly 100 FETs.

TABLE II—Module-Level Signals

BROADCAST CONTROL:	
KFE	KEY FIELD - EVEN
KFO	KEY FIELD - ODD
KTUE	KEY, TAG, UPDATE - EVEN
KTUO	KEY, TAG, UPDATE - ODD
MODE	(Assumes one of the eight states listed in Table 1)
NR	NEXT RESPONSE
DSR	DISABLE STATIC RETRIEVAL
CHAIN CONTROL:	
A_0, A_n	PE ACTIVATION
SS_0, SS_n	SUCCESSFUL SEARCH
RESPONSE	
R	SEARCH RESPONSE, SORT COMPLETION

TABLE III—Internal PE Signals

KF	KEY FIELD
KTU	KEY, TAG, UPDATE
X_i	INPUT FROM } STORAGE LOOP L_i
Y_i	OUTPUT TO } STORAGE LOOP L_i
SS	SEARCH SUCCESSFUL
INT	INTERCHANGE
RF	REPLACEMENT FIELD
RD	REPLACEMENT DATA
A_i	PE_i ACTIVE
SS_i	SUCCESSFUL SEARCH BY PE_j, FOR SOME $j \leq i$
ER	ENABLE RESPONSE
RC	RESET COMPARATOR
RL	RECONFIGURE LOOPS
R_i	RESPONSE FROM PE_i

OPERATION

The intelligent memory provides six primary modes of operation: sort, load/unload, search, retrieve, update and reconfigure. These are described below; performance expressions are given with respect to the following memory parameters:

Loop Size	l	bits
One Bit Transfer Time	t	sec.
Loop Cycle Time	$r=lt$	sec.
Number of Loops	n	

In the paragraphs below, reference is made to signals KFE, KFO, KTUE, and KTUO. Signals KFE (Key Field-Even) and KFO (Key Field-Odd) define the bit positions on which a sort or search is to be made or which are to be updated. There are two such signals because PEs always are divided into two groups or phases, even and odd, which receive any specified group of bits at times separated by half a loop rotation. Signals KTUE (Key, Tag, Update-Even) and KTUO (Key, Tag, Update-Odd) must also be divided into two phases. These signals have multiple uses. During a sort, they are not used; during a search, the part of KTU gated by KF provides the search key, and any "ones" in KTU outside of KF define a tag field, in which the result of the search (signal SS—Search Successful) is deposited. During an update, KTU supplies the update data.

Sort

In sort operation, control inputs to the memory consist of sort key field definition signals KFE and KFO, and broadcast control mode signal equal to SORT<, or SORT>. These control inputs are generated by a memory controller (not described in this paper). The externally specified sort key field may consist of any contiguous block of 1 to l bit positions. The operation of an active processing element PE_i during one complete cycle of its incident loops L_i and L_{i+1} is as follows. The cycle begins when the key field portions of the data records (stored one record per loop) enter PE_i, high order bits first. As long as the two bit streams remain equal, PE_i maintains the thru connection state. The first mixed pair of bits causes PE_i to assume either the thru state or the interchange state, whichever routes the 0 bit to L_i if SORT< is specified, or to L_{i+1} if SORT> is specified, and to maintain that connection state for the remainder of the cycle. Equal key fields cause PE_i to maintain the thru connection for the full rotation. At a given instant of time during the sort, a data record may be distributed over as many as three contiguous loops. The memory correctly performs a stable sort of its contents on the specified key field, either ascending or descending, and provides a done signal at output R upon completion. Maximum sort time is nr/2 sec. Multiple key and non-contiguous key sorts are performed by repetition of the basic single-key stable sort.

The algorithm underlying sort operations of the memory is a circulating, interleaved version of a sorting network construction known as odd-even transposition sort (see Knuth[1] for a description).

Load/unload

To load a file into the memory, one of the two data ports is chosen as load port; the other is appropriately shorted. The memory is cleared. Memory loading is accomplished by invoking the appropriate sort operation with a key field spanning any data field known to be non-zero (perhaps the entire record). The file is presented serially at the input terminal of the load port.

Memory unloading is performed is a similar manner. An appropriate constant, 0 or 1, is presented at the input terminal of the unload port, a sort operation is initiated, and the file appears serially at the output terminal.

Loading or unloading the full memory requires nr sec. Concurrent loading and unloading is possible whenever the load file has a data field whose values are uniformly less than (or greater than) the values assumed by the corresponding data field in the unload file. This condition may be enforced by dedicating a single bit in each file to this purpose. Concurrent load/unload of the full memory requires nr sec.

Time to load and/or unload may be further reduced by partitioning the memory into a number of segments and loading (unloading) the segments in parallel from the same number of data sources (sinks).

Search

In search operation, control inputs to the memory consist of search key field definition signals KFE and

KFO, search key signals KTUE and KTUO, and broadcast mode signal equal to SEARCH=, SEARCH<, or SEARCH>. The externally specified search key field may consist of any contiguous block of 1 to *l* bit positions. The operation of an active processing element PE_i during one complete cycle of its incident loop L_i is as follows. PE_i compares the data on L_i to the key on line KTU during the interval defined by KF, and classifies the data according to whether it is $=$, $<$ or $>$ the key. The classification is registered via the internal state of the comparator (see Figure 3). If this classification agrees with the broadest control mode, signal SS is raised. In either case, after KF drops, SS may be written into loop L_i as a response or tag bit in any field specified by KTU. Average search time is 3r/2 sec., including writing the response bit.

By invoking appropriate sequences of search operations, compound and multiple key searches can be performed. For example, three search operations implement a "between-limits" search. The search operations provided form a functionally complete set. If S_1 and S_2 are any two search operations or sequences of search operations that can be performed by the memory, and if t_1 and t_2 are the corresponding response bits, then logical operators $-$ (NOT), $\wedge$ (AND), and $\vee$ (OR) applied to S_1 and S_2 may be implemented by secondary search operations as follows:

$$S_1 \wedge S_2 \quad t_1t_2 = \text{'11'}$$
$$S_1 \vee S_2 \quad t_1t_2 > \text{'00'}$$
$$-S_1 \quad t_1 = \text{'0'}$$

Direct hardware interpretation of complex data base queries is an important potential application.

Retrieval

There are two classes of retrieval problems to consider: unique response retrieval and multiple response retrieval. In the former case, responding records, if they exist, are known to be unique; in the latter case they need not be.

Unique response retrieval

Immediately following a successful search the responding record is routed directly and persistently to memory output R. Total average response time is 3r/2 sec.

Multiple response retrieval

The memory provides two options for retrievals in this class.

Static retrieval

The name of this option reflects the fact that no inter-loop data movement is involved. After a suitable waiting period T following a search operation, the output ER of the static retrieval control (see Figure 3) is equal to 1 for at most one processing element PE_i. If such a PE_i exists, it is the processing element having smallest index among those performing a successful search. The responding record on loop L_i is routed persistently to memory output R. The waiting period T must be at least as large as the time for a signal to propagate asynchronously through the full chain of static retrieval controls, conservatively taken to be nt sec. Total response time is $3r/2+nt+r=nt+5r/2$ sec. Retrieval of successive responses is triggered by pulsing the NR input.

Static retrieval response time can be substantially reduced by segmenting the memory at retrieval time and polling the memory segments for a response. In this case, total response time for the first retrieval is approximately $nt/s+5r/2$ sec., where s is the number of equal-length memory segments. This is minimized by choosing $s=\sqrt{n}$.

Dynamic retrieval

A sort operation is invoked following a search operation, using the search response bit as sort key. One of the two memory data ports is chosen a priori as the retrieval port, and the appropriate sort operation is used to drive responding records to the output terminal at this retrieval port. Minor additional logic at the retrieval port allows resetting the response bit in a responding record as it arrives at the output terminal, and reloading the modified record via the input terminal at that same retrieval port. This assures that all responding records will be both retrieved and preserved within the memory for subsequent operations. Average total response time for the first retrieval is approximately nr/4 sec.

As in the static case, response time for dynamic retrieval can be shortened by segmenting the memory at retrieval time.

Update

A replacement update may be performed in parallel on all records responding to any sequence S of one or more search operations and tagged, say in bit t. A search operation for t='1' is performed in order to cause the appropriate comparators to assume the = state. This is followed by a mode transition to UPDATE. The update itself is broadcast on lines KTUE and KTUO, along with update field information on lines KFE and KFO. Average total update time is 5r/2 sec.

Reconfigure

In a reconfigure operation the "shape" of the memory is transformed to accommodate various record sizes. With n basic storage loops of size l bits, reconfiguration to n/k loops of size kl bits for any k, $1 \leq k \leq n$, is possible. The activity and phase control (see Figure 3) implements this structural change to the memory. Sort, load/unload, search, retrieve and update operations apply to all memory shapes.

PACKED-LOOP VERSION

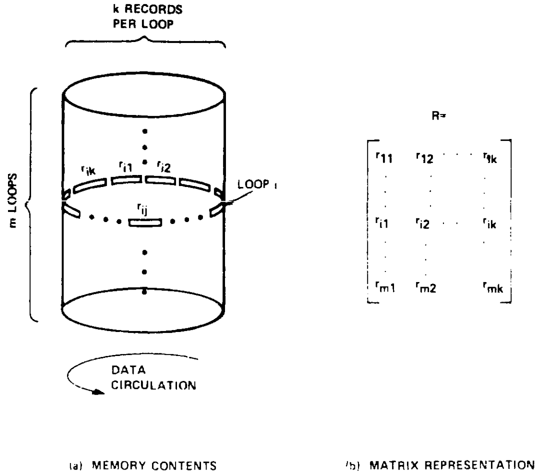
As originally conceived, the memory stores one data record per loop, and is capable of dynamically altering its loop size to accommodate varying record length requirements.

There are cost and technology dependent reasons why alternatives to short (e.g., 32 character) storage loops and on-chip processing elements (PEs) are of interest. For example, while on-chip logic is reasonable for CCD memories, it does not appear feasible for magnetic bubble memories at present. Off-chip logic implies long storage loops in order to maintain reasonable pin counts. Also, longer storage loops imply lower ratios of PE chip area to storage loop area and thus lower memory system cost per bit.

Extension of the intelligent memory architecture to accommodate long loops into which multiple data records are packed is straightforward, except for the sort mode of operation which presents a problem. A new sort algorithm named "gyro sort" was conceived to solve this problem.

Gyro sort

Suppose we have $k \geq 1$ records per loop and $m \geq 1$ loops (see Figure 4) for a total of n=mk records. Let r_{ij} denote the jth record on loop i, and let $R=[r_{ij}]$ be the m×k matrix representation of the memory contents.



(a) MEMORY CONTENTS (b) MATRIX REPRESENTATION

Figure 4—Packed-loop memory

One pass of gyro sort consists of the following two steps:

(1) sort columns
sort the columns of R, say in increasing order from top to bottom

(2) precess rows
for i=1,2,...,m, apply a circular (say right) shift of (i−1) mod k record positions to row i

Figure 5 illustrates the idea of row precession in step (2). In Figure 5(a) we are looking down on a stack of m=7 loops, each divided into k=5 record areas. The top loop is outermost. The numbered circles identify record areas. Figure 5(b) shows the rearrangement of record areas performed by the row precession. The dotted line shows the spiral movement followed by record area 1. This movement, superimposed on the loop rotation movement, motivated our choice of the terms "precession" and "gyro" sort.

We further illustrate gyro sort with a numerical example in matrix form. In this small example, m=6, k=3, and records consist of a single integer. Matrix R_2 below results from matrix R_0 after one pass of gyro sort. Matrix R_1 results from step (1) and R_2 from step (2).

$$\underset{R_0}{\begin{bmatrix} 8 & 3 & 12 \\ 11 & 9 & 2 \\ 5 & 14 & 6 \\ 18 & 1 & 7 \\ 10 & 17 & 13 \\ 16 & 4 & 15 \end{bmatrix}} \underset{R_1}{\begin{bmatrix} 5 & 1 & 2 \\ 8 & 3 & 6 \\ 10 & 4 & 7 \\ 11 & 9 & 12 \\ 16 & 14 & 13 \\ 18 & 17 & 15 \end{bmatrix}} \underset{R_2}{\begin{bmatrix} 5 & 1 & 2 \\ 6 & 8 & 3 \\ 4 & 7 & 10 \\ 11 & 9 & 12 \\ 13 & 16 & 14 \\ 17 & 15 & 18 \end{bmatrix}}$$

Step (1) may be accomplished for general records by the odd-even transposition sort algorithm as provided in the basic intelligent memory architecture. This is subject to a restriction on sort key field position and is discussed further in a later section. The column sort proceeds in parallel on all columns, and requires at most m/2 loop rotations to complete. Step (2) requires a modest amount of additional "precessing logic" and takes one loop rotation to complete if loops can be independently clocked, and several loop rotations otherwise.

Gyro sort consists of k such passes, after which the contents of row i are the records which form the ith block of k records in the final ordered file. Thus a sort of the file into buckets of size k records has been accomplished. There is no particular ordering among the row i records.

Continuing with the example, the second pass yields:

$$\underset{R_3}{\begin{bmatrix} 4 & 1 & 2 \\ 5 & 7 & 3 \\ 6 & 8 & 10 \\ 11 & 9 & 12 \\ 13 & 15 & 14 \\ 17 & 16 & 18 \end{bmatrix}} \underset{R_4}{\begin{bmatrix} 4 & 1 & 2 \\ 3 & 5 & 7 \\ 8 & 10 & 6 \\ 11 & 9 & 12 \\ 14 & 13 & 15 \\ 16 & 18 & 17 \end{bmatrix}}$$

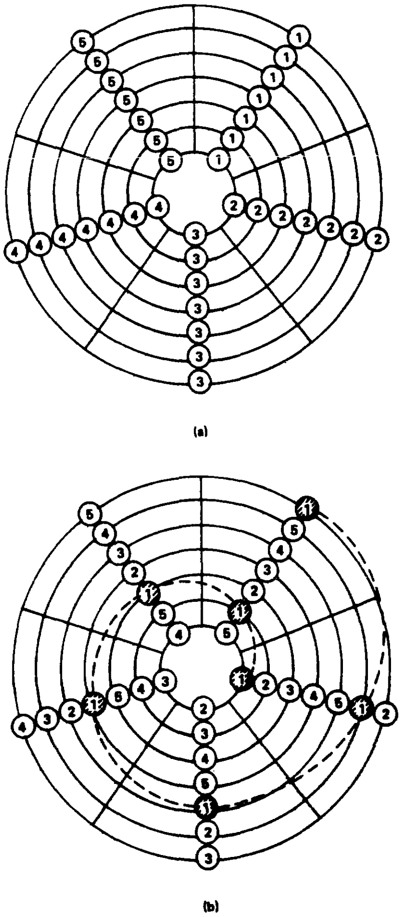


Figure 5—Loop precession

On the third pass we obtain:

$$\begin{bmatrix} 3 & 1 & 2 \\ 4 & 5 & 6 \\ 8 & 9 & 7 \\ 11 & 10 & 12 \\ 14 & 13 & 15 \\ 16 & 18 & 17 \end{bmatrix}$$
$$R_5$$

as claimed. That is, the first three records occupy row 1, the next three row 2, and so on.

We have proven that this claim is true in general.

Precession subsystem

Storage loop contents circulate under control of a loop clock. We distinguish between two types of loop clocking:

(a) Differential loop clocking—the clock signal may be selectively applied to some loops and not to others.

(b) Uniform loop clocking—the clock is applied uniformly to all loops.

Differential loop clocking is feasible in some technologies, e.g., CCD memories, and not in others, e.g., magnetic bubble memories. In the latter case, a rotating magnetic field is typically used to perform uniform loop clocking.

Figure 6 shows the block diagram of a loop precession subsystem. Three elements are represented in Figure 6: a collection of m storage loops $L_1, L_2, \ldots, L_m$, a precession register, and a precession control.

Figure 7 illustrates successive states of the precession register for $m=12$ and a loop packing factor of 5. For each state of the precession register, storage loops L_i for which $Z_i=1$ are advanced one record position relative to storage loops L_j for which $Z_j=0$.

In the case of differential loop clocking, this is accomplished simply by using Z_i to gate the loop L_i clock signal. This is done by precession control cell i.

Figure 8 illustrates precession control cell i for the case of uniform loop clocking, i.e., all storage loops are clocked uniformly. In this case, storage loop L_i is comprised of two sub-loops, the main loop and the auxiliary loop. Total capacity of loop L_i is l bytes; this is apportioned into l-δ bytes for the main loop and δ bytes for the auxiliary loop, where l is an integer multiple of δ.

Precession control is effected by a 2-input, 2-output loop switch at the junction of the main and auxiliary loops which either isolates these sub-loops as in Figures 8(a) and (b) or interconnects them as in Figures 8(c) and (d).

The state of the loop switch in precession control cell i is determined by the logical expression $P \cdot Z_i$ as indicated in Figure 8. When the memory is in precession mode $(P=1)$ the loop switch assumes the isolate state when $Z_i=1$ and the interconnect state when $Z_i=0$. At all other times $(P=0)$ the loop switch maintains the interconnect state.

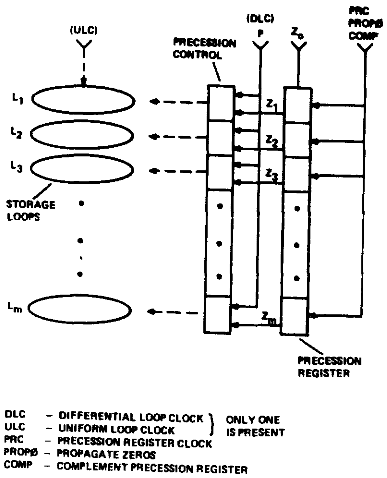


Figure 6—Loop precession subsystem block diagram

Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_7	Z_8	Z_9	Z_{10}	Z_{11}	Z_{12}
0	1	1	1	1	0	1	1	1	1	0	1
0	0	1	1	1	0	0	1	1	1	0	0
0	0	0	1	1	0	0	0	1	1	0	0
0	0	0	0	1	0	0	0	0	1	0	0
1	0	0	0	0	1	0	0	0	0	1	0
0	1	1	1	1	0	1	1	1	1	0	1

Figure 7—Precession register states k=5, m=12

Figure 8 illustrates the phase difference introduced in precession mode between a loop for which $P \cdot Z_i = 1$, (a) and (b), and a loop for which $P \cdot Z_i = 0$, (c) and (d), during a time interval $(l\text{-}\delta)t$, where t is the one-byte shift time (in sec.). The triangular marker in Figure 8 points to an arbitrary but fixed reference position relative to the loop contents. The reference position at time 0 is shown in (a) and (c), and at time $(l\text{-}\delta)t$ in (b) and (d). A phase difference of δ bytes has been introduced.

In (b) the auxiliary loop contents occupy the same position relative to the main loop contents as in (a). A transition to $Z_i = 0$ at time $(l\text{-}\delta)t$ will terminate the differential phasing while preserving the original ordering of stored data. If Z_i is held at 1 for an integral number (say s) of time intervals, each of length $(l\text{-}\delta)t$, then a total phase differential of $s\delta$ is introduced in loop L_i with respect to a loop L_j for which $Z_j = 0$ over that time interval.

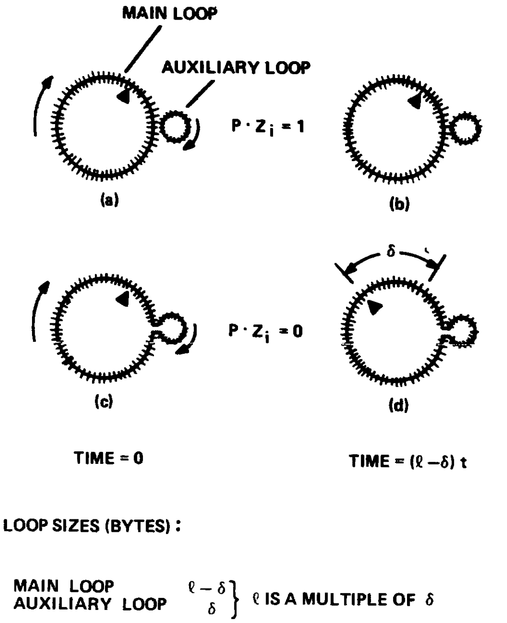


Figure 8—Loop L_i precession control uniform loop clocking

Operation

The same processing element may be used in both the packed-loop and single-record-per-loop memories. This imposes several restrictions on the operation of the packed-loop version vis-a-vis the single-record-per-loop version. The restrictions may be removed at the expense of increasing PE complexity.

In particular, for packed-loop operation, there is no provision for internal memory within each PE to store the current record packing factor k and separate status information about each of k records. Thus, PE actions such as interchanging records, or depositing a tag bit or posting an update following a successful search operation must be completed during one record interval, and not deferred to the next loop rotation. For sorting, this means the sort key field must occur at the head of the record. It may be of any length, however, up to the full record length. For searching, response bits cannot be written ahead of the search key area.

Concerning performance, with a record packing factor of k, gyro sort takes roughly k times longer than with one record per loop, but uses fewer processing elements by a factor of k. A similar tradeoff applies to packed-loop memory search and update operations.

Cost comparison

We assess the cost attainable with the packed-loop intelligent memory as compared with single-record-per-loop. Again we assume n=mk records, m loops, and k records per loop. We let q be the record size in bytes and $l=kq$ be the loop size in bytes.

Suppose we let a_s denote the chip area (say in nanoacres) for one byte of storage and a_p the chip area for one processing element (PE). Then the total chip area for the packed-loop case is

$$nqa_s + ma_p.$$

The *PE-overhead,* defined as the ratio of total PE area to total storage area is

$$\frac{1}{k}\frac{a_p}{qa_s}.$$

For the one-record-per-loop organization, total chip area is

$$nqa_s + na_p.$$

This assumes that a_p is the same in both cases. PE-overhead is

$$\frac{a_p}{qa_s}.$$

Thus the packed-loop organization cuts overhead by a factor of k, and reduces total chip area.

We define the PE-complexity α as follows:

$$a_p = \alpha \cdot a_s.$$

Thus a PE requires chip area equal to that of α bytes of storage. Then the ratio of one-record to multiple-record total chip areas is:

$$\frac{nqa_s+n\alpha a_s}{nqa_s+\frac{n}{k}\alpha a_s}=\frac{q+\alpha}{q+\frac{\alpha}{k}}$$

The PE-complexity α is fixed by the PE design and the technology used. A conservative estimate for α using CCD technology is 18.75 (equivalent to 150 bits of storage). Figure 9 is a plot of cost ratio (one-record-per-loop to multiple-record-per-loop) as a function of the record length q and the packing factor k, using $\alpha=18.75$. Figure 10 is a similar plot, using $\alpha=187.5$ as a representative figure for off-chip MOS PEs and magnetic bubble storage.

SUMMARY

We have described the structure and operation of an intelligent memory subsystem which combines the economy of new serial storage technology with the performance of highly parallel execution of data processing primitives. For a CCD/MOS realization with fundamental loops of size 256 bits, we estimate subsystem cost at 0.5¢ per bit; performance figures are given below, alongside rough comparable estimates for a single CPU and RAM. A memory-resident file of 32,000 32-character records is assumed in both cases. A 10 MHz shift rate is assumed for the intelligent memory; a 1 μs. cycle time is assumed for the CPU-RAM.

	Intelligent Memory	CPU-RAM
LOAD/UNLOAD	1 s.	1 s.
SORT	0.5 s	15 s.
SEARCH	40 μs.	75 ms.
SEARCH AND RETRIEVE	100 μs.	75 ms.
SEARCH AND UPDATE	65μs.	75 ms.
RECONFIGURE	4 ms.	—

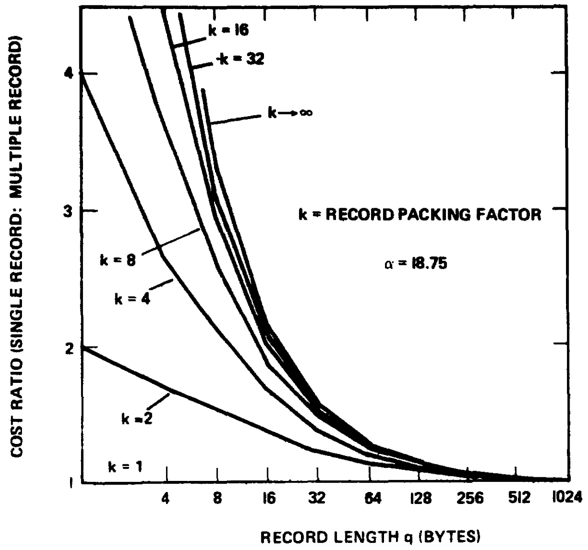


Figure 9

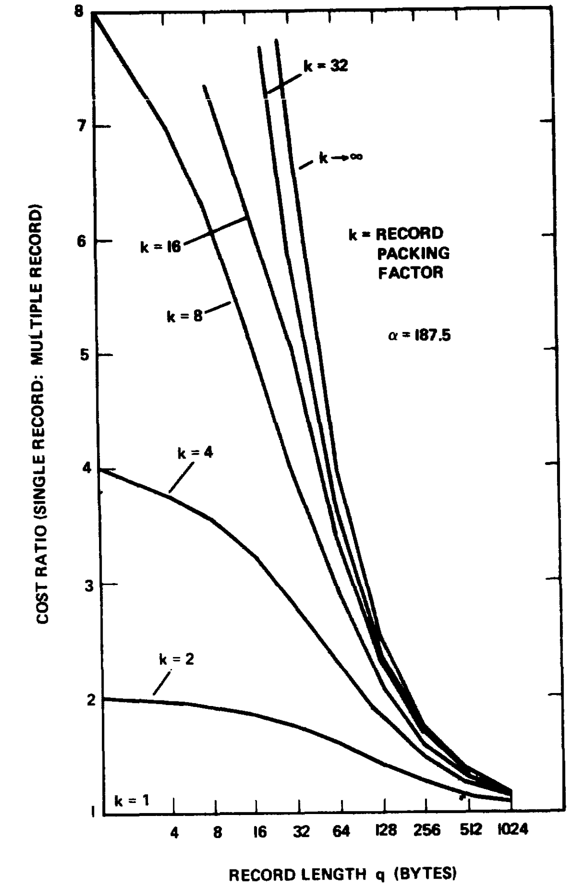


Figure 10

We have shown how performance can be traded for economy by extending the memory architecture to accommodate packed-loop operation. This is aimed at realizations using long storage loops in which lower subsystem cost (we estimate 0.05¢ per bit for magnetic bubble storage and off-chip MOS PEs) may make large capacity intelligent storage subsystems feasible for future data-oriented computer systems.

ACKNOWLEDGMENT

The authors would like to acknowledge the contributions of T. Bonn to this work. The idea of extending the intelligent memory to operate with multiple data records per loop, and the idea of precessing loops as part of a packed-loop sort strategy are due to him.

REFERENCE

1. Knuth, D. E., "Sorting and Searching" *The Art of Computer Programming*, Vol. 3, Addison-Wesley, 1973.

Approaches to computer reliability—Then and now*

by ALGIRDAS AVIŽIENIS
University of California
Los Angeles, California

*To the ingenious designers and programmers
of the first generation
Who made their machines work in spite of
most contrary components
And inspired their successors to continue
striving for reliable computing*

ABSTRACT

Approaches to the attainment of reliable computer operation are considered in this paper. The goal is to assure correct execution of programs using less than perfect components. The discussion includes design methodology, fault classification, redundancy techniques, reliability modeling and prediction, and examples of fault-tolerant computers. The last section identifies some relationships between reliability methods for hardware and for software.

HISTORICAL PERSPECTIVE

The problem of reliability has confronted both the designers and the users of computing systems since the building of the first computers in the 1940's. First-generation digital computers used large numbers of vacuum tubes, relays, and other electromechanical devices which were notably failure-prone. Various methods of failure detection and recovery were incorporated in the hardware of these machines. For example, duplicate arithmetic-logic units were used in the EDVAC and UNIVAC computers; various error-detecting codes were used in many others, including RAYDAC, IBM 650, NORC, etc.[1]

The advent of transistor technology in the second generation led to a very large improvement in component reliability. This improvement, in turn, led to a deemphasis of failure detection techniques in the hardware. The remaining exceptions were parity checking and related techniques in storage and I/O equipment. All failures, however, were not eliminated, and the absence of hardware checking led to the rapid development and extensive use of diagnostic programs. The diagnostics were employed to perform periodic checkouts of computers and to assist maintenance specialists by identifying failed parts during repairs. Widespread use of diagnostics began in the late 1950's and has continued into the present, with microdiagnosis largely superseding diagnosis in the middle to late 1960's.

Diagnosis-aided manual repair, however, proved in many cases to be an insufficient solution because of at least three reasons: (1) the unacceptability of the delays and interruptions of real-time programs caused by manual repair action; (2) the inaccessibility of some systems to manual repair; and (3) the excessively high cost of lost time and of maintenance in many installations.

Since the early 1960's the scope of computer applications has steadily expanded, encompassing numerous areas of critical importance. These applications include real-time control of communication and transportation systems, manned space flights, automated factories and power plants. At the present, use of computers is being considered for the monitoring of critically ill patients in hospitals. The reliability requirements for computers in such applications far exceed the requirements established for the computing systems of the 1950's and 1960's. The expected great benefits of computer use are balanced against the potentially disastrous costs of their failure.

Another relevant development of the past decade has been the wide distribution of computing systems throughout the entire planet and their use in space. Instead of being concentrated in a limited number of population centers, computers are now performing important, and even critical tasks in many locations that are remote from the service and repair facilities and personnel. Computers have been employed in space

* This work was supported by the National Science Foundation, Grant No. DCR 72-03633 A03

vehicles orbiting the Earth and the Moon, or traveling to the other planets of the solar system. In these applications fully automatic detection of faults, program restart, and self-repair are either absolute requirements, or economic necessities in order to provide reliable computing at an acceptable cost or risk to the user.

In the most general sense, reliable computing means "the correct execution of a specified set of algorithms", and encompasses all the following elements:

—correctness and completeness of software specifications;
—testing and verification (proofs) of programs;
—elimination of hardware design errors;
—continued correct execution of programs and protection of data in the presence of hardware failures;
—protection of the computing system against error-induced disruption or deliberate invasion of programs and data.

Attempts to meet the requirement for reliable computing, especially when external assistance by maintenance specialists is too slow, too costly, or not available, have utilized the two complementary approaches of *fault-intolerance* and *fault-tolerance*.[2] These approaches are applicable to all parts of the computing system, including its hardware elements, microprograms, system programs, and user programs.

In the "fault-intolerance" approach the reliability of computing is assured by *a priori* elimination of the causes of unreliability, i.e., of faults. This elimination takes place before the normal computing process, and the resources that are allocated to attain reliability are spent on perfecting the system prior to its field use. Since in practice it has not been possible to assure complete *a priori* elimination of all causes of unreliability, the goal of fault-intolerance is to reduce the unreliability (expressed as the probability of system failure over the duration of the specified computing process) or the unavailability to an acceptably low value. To supplement this approach, manual maintenance procedures must be devised which return the system to an operating condition after a failure. The cost of providing readily available maintenance and the cost of the disruption and delay in the computing process also are parts of the overall cost of using the fault-intolerance approach.

In the "fault-tolerance" approach the reliability of computing is assured by the use of protective redundancy. The causes of unreliability are expected to be present and to induce errors during the computing process, but their disrupting effects are automatically counteracted by the redundancy. Reliable computing is made possible despite the remaining program and hardware design errors, hardware failures, and external interference with computer operation. The resources allocated to attain reliability are spent on protective redundancy. The redundant parts of the system (both hardware and software) either take part in the computing process or are present in a standby condition, ready to act automatically to preserve its undisrupted continuation. In contrast, we note that the maintenance procedures in a fault-intolerant system are invoked after the computing process has been disrupted, and the system remains "down" for the duration of the maintenance period.

It is evident that the two approaches are complementary and that the resources allocated to attain the required reliability of computing may be divided between fault-tolerance and fault-intolerance. Experience and analysis both point to the conclusion that a balanced allocation of resources between the two approaches is most likely to yield the highest reliability of computing. An overview of past practice shows that fault-intolerance has been the dominant choice in both hardware and software in the 1950's and 1960's. In recent years the fault-tolerance approach has been making significant inroads in hardware system design; its application in software has remained very limited. The cost of redundancy has been the main argument against the use of fault-tolerance techniques in computer systems. The evolution of component technology into large-scale integration, the decreasing cost of mass-produced hardware elements, the very high cost of software testing, and the increasing reliability requirements all favor increasing use of fault-tolerance in computer systems of the future. The resistance to its wider use frequently originates with the practitioners of the current fault-intolerance and manual maintenance methods.

DESIGN METHODOLOGY FOR RELIABLY OPERATING COMPUTERS

This section and the subsequent sections address the issue of providing undisrupted computing while using less than perfect hardware components. The remaining aspects of reliable computing remain outside the scope of this paper.

Unreliable operation is caused by imperfections in the physical implementation of the computer's logic structure. Reliability theory defines the reliability $R(T)$ of a system as the probability of its correct operation up to the time $t=T$, given that the system was operating correctly at the starting time $t=0$. Computers differ from other systems because in their case "correct operation" means the correct execution of a set of programs and protection of data, rather than the continued functioning of a set of physical components of the system. It is the purpose of this section to present those aspects of computer system design that are specifically directed toward the elimination or tolerance of imperfections (called "faults") in the components of the system. It is to be noted that we discuss correct *execution* of a given set of programs and do not include the questions of correctness of the programs, completeness of their specification, and of accuracy of the algorithms, which remain separate fields of study.

The architects and the users originate the sets of programs and data, the definitions of required operations, the time limits for program execution, and the storage requirements. The objective of the designer is to raise the reliability (i.e., the probability of correct execution of these programs) or availability to an acceptably high value, given that *operational faults* may occur during execution. Such faults are caused by three classes of physical events that affect the hardware of the system:

—permanent failures of hardware components;
—intermittent malfunctions of components;
—external interference with computer operation.

As discussed previously, two complementary approaches have been employed to attain satisfactory reliability. Fault-intolerance is the approach that aims to reduce the probability of occurrence of the first fault during a specified time interval to an acceptably low value. In the "pure" fault-intolerance approach the system is designed without redundancy, and every component of the system must function correctly in order to assure correct program execution. The procedures which lead to the attainment of reliable "fault-intolerant" systems are:

—the most reliable components are acquired within the existing cost and performance constraints;
—proven techniques are employed for the interconnection of components and assembly of subsystems;
—the system is packaged to screen out the expected forms of external interference;
—quantitative prediction of system reliability is made using known or predicted failure rates for the components and interconnections.

In the "purely" fault-intolerant (i.e., non-redundant) design, the probability of fault-free hardware operation is equated to the probability of correct program execution. Such a design is characterized by the decision to invest *all* the reliability resources into procurement of high-reliability components and refinement of assembly and packaging techniques. An alternative to the "purely" fault-intolerant approach is offered by the use of various forms of redundancy to attain fault-tolerance.[3] This approach increases reliability by the use of design techniques that allow faults to occur without disrupting the continued correct execution of the programs. Fault-tolerance does not entirely eliminate the need for reliable components; instead, it offers the option to allocate part of the reliability resources to the inclusion of redundancy. The goal of a fault-tolerant design is either a reliability (or availability) prediction that cannot be attained by the purely fault-intolerant design, or a reliability (or availability) prediction that matches the purely fault-intolerant design at a lower overall cost of implementation.

A fault-tolerant computer system must possess the following attributes:[4]

—Its description includes a set of components (hardware) and a set of programs (software).
—It is either initially free of design faults or it is protected against their disruptive effects during program execution.
—It executes the set of programs correctly in the presence of operational faults.

The first attribute stresses the fact that the ability of a computer to continue operating correctly in the presence of operational faults depends not only on the properties of the hardware, but also on the nature of the software, including both the system programs and the user programs. For example, the ability to recover from the errors caused by transient faults frequently depends on special restart features incorporated in the system software as well as on proper partitioning and state vector storage of user programs.

The second attribute requires that design faults should be eliminated from both hardware and software prior to the initiation of the computing process. As an alternative, protective features for the detection and circumvention of design faults in both hardware and software must be incorporated to make the system fault-tolerant. Design faults are caused by errors made during the translation of the original specifications into operational forms, that is, into assemblies of components and of machine language instructions. They are eliminated by validation of the hardware and software designs prior to their operational use. Since complete *a priori* verification cannot yet be assured, computers need protective provisions to detect and circumvent abnormal conditions encountered during operation which may be symptoms of remaining design faults. A completely fault-tolerant operation is attained either when all design faults are eliminated from the system, or when complete protection against remaining design faults is incorporated.

The third attribute of a fault-tolerant computer postulates correct execution of the entire set of programs in the presence of operational faults. Program errors that are caused by faults in the hardware can be avoided or corrected by means of protective redundancy. Protective redundancy may be introduced in three forms:

—additional hardware (hardware redundancy);
—additional software (software redundancy);
—repetition of operations (time redundancy).

These redundant features would not be needed in a fault-free computer; that is, their deletion does not affect computer performance in the absence of operational faults. Given that the faults will occur in the hardware, the redundant features provide a fault-tolerant computing system which carries out its programs correctly in the presence of operational faults. Partial fault-tolerance (also called "fail-soft operation" or "graceful degradation") occurs when operation continues, but one or more programs are not correctly executed in the specified time.

Research results and design experience lead us to suggest that the introduction of protective redundancy can be accomplished by following a systematic procedure:[4]

(1) Performance requirements are established and system architecture is specified with the initial assumption that operational faults will not occur (the "fault-intolerant" design).
(2) Classes of operational faults that are to be tolerated in the design are identified, and the extent of tolerance is specified for each class of faults.
(3) Cost-effective methods of protective redundancy (time, hardware, software) are chosen to cover every class of faults identified above, and system architecture is modified to incorporate the redundancy.
(4) Analytic or experimental techniques are employed to estimate the extent of fault-tolerance that is provided by the protective redundancy.
(5) Checkout methods are devised to test all redundancy features. Where applicable, fault-tolerance is extended to effect automatic maintenance of peripheral systems that are connected to or controlled by the computer.

Design experience has shown that several iterations of (3) and (4) may be necessary to arrive at a satisfactory fault-tolerant system architecture. The following sections discuss the techniques for the implementation of (2) to (5) and illustrate their use in recent computer systems and in proposed designs.

CLASSES OF OPERATIONAL FAULTS

An operational fault is the deviation of one or more logic variables in the computer hardware from their design-specified values. Faults are caused by failures, which are physical changes in the hardware of the computer. Hardware failures are of three types: "solid" component failures, "intermittent" component malfunctions, and externally caused interference with the operation of the computer. The immediate symptom of any hardware failure is a fault. The fault often causes an error in the program being executed by the computer: either an instruction is not executed correctly, or an incorrect result is computed. Both types of errors may be caused at once by some faults.

A systematic approach to the choice of redundancy techniques in computer design begins with a classification of faults and identification of those classes which are expected to occur in the system being designed. Three useful dimensions for the classification of faults are:[4]

—*duration:* transient (intermittent) vs. permanent (solid);
—*extent:* local (single) vs. distributed (related multiple);
—*value:* determinate ("stuck") vs. indeterminate (variable).

In the design methodology the "permanent vs. transient" classification appears to be most fundamental because the two classes usually need different recovery methods. A program restart is sufficient to correct errors caused by transient faults, while replacement or reconfiguration of hardware is needed to eliminate permanent faults from the system. The classifications according to extent and according to value are applicable to both transient and permanent faults.

The extent of a fault specifies how many logic variables in the hardware are simultaneously affected by the fault which is due to a single failure event. *Local* (single) faults are those that affect only single logic variables, while *distributed* (related multiple) faults are those that affect two or more variables, one module, or an entire system. The physical proximity of logic elements in contemporary MSI and LSI circuitry has made distributed faults much more likely than in the discrete component designs of the past. Distributed faults are also caused by single failures of some critical central elements in a computer system, for example: clocks, power supplies, data buses, switches for computer reconfiguration, etc.

The value of a fault is *determinate* when the logic values affected by the fault assume a constant value ("stuck on 0" or "stuck on 1") during its entire duration. The fault value is *indeterminate* when it varies between "0" and "1", but not in accord with design specifications, during the duration of the fault.

It is important to observe that a precise description of fault extent and fault value can only be made at the source of the fault, that is, at the point at which the hardware failure event has actually taken place. The introduction of one or more faulty logic variables into the computing process will often lead to different or more extensive fault symptoms downstream from the point of failure. For example, a "stuck on 1" local determinate fault on the input to a two-input "Exclusive-Or" gate will cause the output variable of the gate to appear as a local indeterminate fault. Furthermore, if this output is supplied as an input to several other gates, the set of output variables of these gates will appear as a distributed indeterminate fault.

Ambiguity is avoided when the term "fault" is restricted to the change in logic variable(s) at the point of the actual hardware failure. The fault-caused changes of logic variables which are observed (because of faulty inputs) on the outputs of correctly functioning logic elements are symptoms of the fault and will be called errors. This distinction establishes a cause-effect sequence as follows:

(1) The *failure* which is a physical event, causes a *fault,* which is a change of logic variable(s) at the point of failure.

(2) The *fault,* in turn, supplies incorrect input(s) to the computing process and causes an *error* to be produced by subsequent operation of failure-free logic circuits.

The preceding discussion makes it evident that the detectability of a fault depends not only on its type, but also on the distance (in terms of computing operations) from the point at which the fault occurs to the point at which checking (fault-detection) is performed. A local determinate fault may cause an extensive error pattern to appear at a point that is several computing steps (in time, space, or both) removed from the fault itself.

METHODS OF PROTECTIVE REDUNDANCY

The key to successful application of protective redundancy is the systematic and balanced selection of suitable methods of its three forms: *hardware* (additional components), *software* (special programs), and *time* (repetition of operations). This section reviews the basic methods of these forms of redundancy.

Hardware redundancy

Hardware redundancy includes the components that have been introduced into the system in order to provide fault-tolerance. As long as faults do not occur, all these components can be deleted without diminishing the computing power of the system. The techniques of introducing hardware redundancy may be divided (on the basis of terminal activity of modules) into two categories: *static* redundancy and *dynamic* redundancy.[5]

The static redundancy method is also known as "masking" redundancy, since the redundant components are employed to mask the effect of hardware failures within a given hardware module, and the terminal activity of the module remains unaffected as long as the protection is effective. The static technique is applicable against both transient and permanent faults. All redundant copies of an element are permanently connected and receive power. Component failures and logic faults are masked by the presence of other copies of the same element. The fault masking occurs instantaneously and automatically; however, if the fault is not susceptible to masking and causes an error, a delayed recovery is not provided.

The original study of the use of static redundancy at the logic element level is due to John von Neumann.[6] He considered transient malfunctions of individual logic gates and showed that arbitrarily high reliability would be attained with high orders of redundancy. Moore and Shannon[7] applied the static redundancy principle to relay contact networks. In practical applications the order of redundancy has to be as low as possible in order to make the cost acceptable to the user. Two forms of static redundancy have been used in practice: replication of individual electronic components in the Orbiting Astronomical Observatory and triple modular redundancy (TMR) with voting in the SATURN IV and V guidance computers.[8] Several other variants of static redundancy have been studied but were not employed in practice because of excessive cost or the need for practically unrealizable special components. It is essential to note that static redundancy is based on the assumption that failures of the individual copies are independent. When related failures take place, the protection by redundancy is lost. For this reason static redundancy (especially at the component level) is not applicable within integrated circuit packages, in which individual components are in close proximity and failure phenomena frequently affect several adjacent components.

In the dynamic redundancy approach fault-caused errors are allowed to appear at the terminals of a module. Fault-tolerance is implemented by two consecutive actions. First, the presence of a fault is detected, then a recovery action either eliminates the fault, or corrects the error. If human assistance is completely eliminated, dynamic redundancy (usually with software support) results in self-repair of a computer system. Limited, that is human- and software-assisted, use of dynamic redundancy techniques in computer hardware has been very extensive.[1,3,5] The most common example is the use of parity to detect errors in data transmission and storage. Important early examples of extensive dynamic redundancy with software and human support are the ESS systems.[9,10] Probably the first operational computer with full self-repair provisions is the JPL-STAR computer.[11]

The application of dynamic redundancy to a computer architecture requires that a number of decisions should be made in the functional design stage. The design choices include: level of modularization, fault-detection hardware, type of recovery action, "hard-core" protection, forms of intermodule communication, validation of inputs, and interfaces with system software.[2] The use of dynamic redundancy has been somewhat inhibited because of the need for an early commitment to it in the hardware design process. In contrast, static redundancy (and software redundancy, as well) can be applied to an existing non-redundant design.

Software redundancy

Software redundancy includes all additional programs, program segments, instructions, and microinstructions which would not be needed in a fault-free computer.[2] They provide either fault-detection or recovery in fault-tolerant computer systems, very frequently in conjunction with dynamic hardware redundancy. Three major forms of software redundancy are:

- —multiple storage of critical programs and data;
- —test and diagnostic programs at various program and microprogram levels;
- —fault-tolerance features of the executive program which implement program restarts and interface with the dynamic hardware redundancy.

Combinations of all three forms are found in most modern fault-tolerant computers.

Compared to hardware redundancy, an advantage of software is the ability to superimpose fault-tolerance features after the hardware has been designed. This allows the design of fault-tolerant systems using non-fault-tolerant 'off-the-shelf' hardware. Another advantage is the relatively easier modification and refinement of these software features after their introduction into the system. The main disadvantage of software redundancy is the difficulty of assuring that the software features will be able to function correctly after the occurrence of a fault and that they will be invoked sufficiently early, that is, before the fault-caused errors have irrevocably disrupted the programs or mutilated the data base. Other disadvantages include the relatively high cost of generating the required software, the storage requirements, including the need to tolerate failures of memories holding the software, and the difficulty of estimating and proving the completeness or the adequacy of the software redundancy features. It must be stressed that dynamic hardware redundancy and software redundancy are not mutually exclusive in practice.[12] A system with all-out emphasis on self-contained dynamic hardware techniques still needs cooperation from the executive program to complete some recovery actions. Conversely, an all-software controlled fault-tolerant system has high risks of excessive delays in initiating recovery without at least some hardware methods for fault-detection. It also needs redundant storage modules and hardware protection of critical decision-making logic. Combinations of software and hardware redundancy are employed in most fault-tolerant systems, but they differ in the choice of the point in the detection and recovery sequence at which software takes over control.[9,11-14]

Time (execution) redundancy

This form of redundancy consists of repeating or acknowledging machine operations at various levels: micro-operations, single instructions, program segments, or entire programs. It is usually employed together with dynamic hardware and software redundancy techniques. Two distinct goals of time redundancy are:

- —fault detection by means of repeated execution or acknowledgments;
- —recovery by program restarts or operation retries after fault detection or reconfiguration has occurred.

The repeated execution of a program is probably the oldest form of fault detection. While suitable to detect errors due to transient faults, it is limited by the fact that consistent errors will be produced by permanent faults, and comparison will fail to reveal the same error in the results. The use of retransmission and of other forms of acknowledgments ("handshakes") has been extensively used in general purpose systems, especially for error detection in secondary storage, channels, and I/O devices.[13,22]

Another common use of time redundancy is found in the identification and correction of errors caused by transient faults, and in program restarts after a hardware reconfiguration.[2] This is accomplished by the repetition after error-detection or 'rollback' of single instructions, segments of programs, or entire programs. While single-instruction retries are transparent to the programmer, longer rollbacks require programming constraints as well as protected storage for the rollback address and for the state vector, including its double-buffering. "Singular" events in a computer are program-controlled events which should not be repeated as part of a program rollback operation, for example, real-time output commands which initiate irreversible actions in the system under computer control. The potential damage makes it imperative that the provision for handling of singular events should be incorporated in rollback procedures.[11,14]

Checkout and extension of fault-tolerance features

The introduction of redundancy poses the problem of verifying that the redundant parts are ready to be used when faults occur. Implementation of checkout encounters difficulties in systems with static hardware redundancy, especially in component-redundant systems.[7,8] Dynamically redundant systems are inherently better suited for redundancy checkouts, since they possess extensive fault-detection and permit sequential switching-in of spare modules to be tested.[11] One critical requirement of checkout is the systematic verification that all fault-indicating signals are operational. Self-checking logic[15] is suitable for this purpose. In software-controlled fault-tolerance this function is carried out by a special fault-signal-test instruction.[16]

The techniques of fault-tolerance can be systematically extended beyond the boundaries of the fault-tolerant computer to effect automatic maintenance of various peripheral systems which communicate with the computer. The methodology of extending fault-tolerance consists of the development of fault-tolerant interfaces, introduction of fault-detection methods in the systems outside the computer, and programming of recovery sequences to be executed by the computer. A case study of the application of these techniques in a spacecraft system is presented in Reference 17. In commercial general purpose systems, the reverse process has taken place. Because of the relatively high unreliability of peripheral mechanical devices, fault-tolerance began at the peripherals and only later was brought into the CPU and main memory.[28]

FAULT-TOLERANT SYSTEMS

The currently existing and proposed fault-tolerant computer systems may be conveniently classified according to the method used to control the recovery. *Hardware-controlled* systems use dedicated hardware which collects fault indications and initiates recovery. While recovery control may be transferred to software after its operability has been assured, it is completed automatically (without external aid). *Software-controlled* systems depend on special programs to interpret fault indications and to carry out the automatic recovery procedures. *Manually-controlled* systems require the participation of a maintenance operator in the completion of recovery; they are not fault-tolerant in the full sense of the word, although they may employ many fault-tolerance techniques.

Hardware-controlled recovery

This approach depends on special hardware to carry out fault detection and to control the initial recovery procedure. After the procedure has established the existence of an operational software system, the completion of recovery is usually transferred to software control. It is evident that further software systems may be superimposed on the hardware-controlled design, leading to a multilevel recovery procedure. A special case of hardware-controlled recovery is found in statically-redundant systems in which faults are masked by redundant hardware, and are totally invisible to the software. Two examples of such systems are the OAO data processor which used component redundancy and the CPU of the SATURN V guidance computer, which used TMR protection.[8] A separate software-controlled recovery system is needed in statically-redundant systems if they are to continue operating in reconfigured mode after the first fault that escapes the masking effect and affects the software.

Dynamically redundant systems usually depend on a dedicated hardware module that gathers fault signals and initiates recovery. Different uses of duplexing and hardware-controlled switchover techniques are found in the memory, power supply, and peripheral units of SATURN V computer in combination with a TMR-protected serial CPU unit.[8] Separate fault-detection and switchover-control units were used for every functional unit. Probably the first operational computer with fully hardware-controlled dynamic redundancy was the experimental JPL-STAR computer.[11] Intended for self-contained multiyear space missions, this computer employs a special Test-And-Repair-Processor (TARP) module to control recovery and self-repair. Software assistance is invoked only to perform memory copying and to resume standard operation after self-repair.[14] The French MECRA computer is another early experimental design.[18] A few other hardware-controlled system designs that have not reached operation have been described in recent literature.[12,16]

The principal advantage of hardware-controlled recovery systems lies in their independence of the operation of any software immediately after the fault has occurred. The recovery process is transferred to software only after its ability to operate has been assured. The relatively late appearance of such systems may be attributed to the need to introduce the recovery module into the design at its inception, thus requiring an early commitment to the hardware-controlled approach.

Software-controlled recovery

In contrast to the previous class, the software-controlled systems depend on special software to initiate recovery action upon the detection of a fault. Fault signals are obtained by both hardware and software methods, for example: parity checkers, comparators, power level monitors, test programs, reasonableness checks, etc. The main limitation of these systems is the need for the recovery software to remain operational in the presence of faults, since recovery cannot otherwise be initiated.

A significant advantage of this approach is that existing 'off-the-shelf' hardware system modules may be used to assemble fault-tolerant organizations. These modules contain various forms of hardware fault-detection, which usually are supplemented by further software methods. For this reason software-controlled systems have appeared earlier and are currently being used in numerous applications requiring high reliability and availability. While every modern operating system incorporates some recovery features, the present paper will be limited to selected illustrations of historically important and advanced systems.

An important early design of the 1950's with complete duplication and extensive recovery provisions was the SAGE system.[19] The IBM System/360 architecture contains very complete provisions for multi-system operation in order to attain high availability, reconfiguration, and fail-soft operation.[21] An early example of a multi-system which includes further extensions of the System/360 design is the IBM 9020 multiprocessing system for air traffic control applications.[13] Noteworthy are the operational error analysis program and the diagnostic monitor of the 9020. The recent IBM System/370 hardware incorporates automatic retry of CPU operations, error coding to correct single-bit errors in processor and control storage, and I/O retry facilities. The software provides recovery management support routines, I/O and channel error recovery, checkpoint/restart facilities, microdiagnostics and on-line diagnostics of I/O device errors.[22] An interesting illustration of extensive use of backup storage and dynamic reconfiguration in a general-purpose time-shared system is found in the MIT Multics System.[23] Another experimental system for high-availability performance in an interactive time-shared environment is PRIME.[24] The Pluribus is a minicomputer/multiprocessor system with extensive fault-tolerance provisions which serves as a switching node in the ARPA Network.[20]

Another direction of software controlled system development is in aerospace applications. The principal illustrations of this approach are the SIFT design,[25] the RCS system,[26] and the C.S. Draper Laboratory modular system.[27] One more area of application which requires fault-tolerant operation and very high availability for several years of continuous operation is the control of electronic switching systems for telephone exchanges. These systems usually employ manual repair by replacement of a failed part as the last (offline) step of the recovery procedure, while maintaining normal operation by means of the remaining system modules. A well documented illustration is found in the ESS systems of Bell Laboratories.[9,10] ESS systems employ a variety of hardware techniques (duplication, matching, error codes, function monitors) and special software (check routines, diagnostics, audits) as well as software and hardware emergency procedures when normal recovery action does not succeed.

Fault-tolerant memories and processors

Besides the complete systems discussed above, significant efforts have been carried out in providing fault-tolerance for storage subsystems. This is especially true for secondary and mass storage which has been characterized by relatively low reliability in the past. Representative error coding applications include the use of codes for error control in data communications, magnetic tape units, disc files, primary random access storage, and a photo-digital mass store.[28] Single-error correcting codes are used in the control storage of ESS No. 1 and the main and control storage of IBM System/370 computers.[9,22] Error-correcting codes have been shown to provide a very effective method for fault-tolerance in the storage medium, and remaining problems are concentrated in providing fault-tolerance in the memory access and readout circuitry.

Recent studies have considered the problem of fault-tolerance in associative memories and processors.[29] In general, processor fault-tolerance has been provided by duplication and reconfiguration at the system level. Some investigations have been conducted in the use of arithmetic error codes as the means for error-detection for processor faults[30] and an experimental processor has been designed and constructed for the JPL-STAR computer.[11] The increasing availability of microprocessors makes further emphasis on duplication very likely, although error-detecting codes remain a convenient method for the identification of the faulty processor in a disagreeing pair.

RELIABILITY MODELING AND PREDICTION

The initial choice of redundancy techniques requires verification that the redundant system possesses the expected fault-tolerance. Insufficiencies of the original design may be uncovered, and the design can be refined by changes or additions of various forms of redundancy. The process is repeated until a fully satisfactory design is attained. The principal quantitative measures are reliability[31,32] (with respect to permanent faults), survivability[4,37] (with respect to transient faults), and availability.[32] Two approaches to the prediction of fault-tolerance are:

—the *analytic* approach, in which fault-tolerance measures of the system are obtained from a mathematical model of the system, and

—the *experimental* approach, in which faults are inserted either into a simulated model of a system, or into a prototype of the actual hardware, and fault-tolerance measures are estimated from statistical data.

A quantitative reliability prediction for the computer being designed requires numerical failure rates for the components. When technologies which are under development are to be used, the failure rates for currently used components need to be extrapolated to the new choice of component technology. It is important to recognize that different failure rates or distributions may apply to failures causing distributed faults. The principal measure of fault tolerance with respect to permanent faults is the reliability $R(t)$, which is a function of the failure rates and directly predicts only the probability of hardware survival. Fault-tolerance is attained only if correct program execution is maintained by the surviving hardware; for this reason transient faults must also be considered. A very common quantitative measure has been the MTBF (mean time between failures), defined as MTBF$=\int_0^\infty R(t)\,dt$. Given the non-redundant reliability $\mathrm{R}(t)=e^{-\lambda t}$, we have MTBF$=1/\lambda$, and the comparison of the MTBFs directly compares the total failure rates (λ) of the competing systems. When redundancy is introduced, the reliability function $R(t)$ is a polynomial in $e^{-\lambda t}$ and the $R(t)$ curves of systems being compared may have crossover points. Then the area under the $R(t)$ curve does not indicate which system is better at a given time, and the MTBF may become a misleading measure. Given a fixed 'mission time' T, the comparison of two or more systems requires only the values of $R(T)$ in order to select the best system. If a fixed mission time is not available, the time interval during which the reliability remains above a given value serves as a convenient comparison measure.[31]

It is essential to note that reliability modeling remains useful even if definite numerical failure rates and mission times are not available, since it permits the comparison of many alternate designs using normalization with respect to the (failure rate × mission time) product λT.

Reliability models

The class of *static* reliability models is suitable for the reliability prediction of systems with static hardware redundancy. The non-redundant system or its element is usually assumed to have the reliability $R(t) = e^{-\lambda t}$. The redundant elements are assumed to be permanently connected, and to fail statistically independently. They have the same failure rate and are instantaneously available to perform the masking of a failure with unity probability of success. Under these assumptions, the reliability of a redundant system is obtained as the sum of the reliabilities of all distinct configurations (including none or some failed parts) that do not lead to system failure. For example, given the simplex (one system) reliability R, the reliability of a duplicated system is $R(duplex) = R^2 + 2R(1-R)$. In general, reliability models of static redundancy are found in standard handbooks and textbooks of reliability theory and are used for reliability analysis of various physical systems.[32]

Dynamic redundancy requires the consecutive actions of fault detection and recovery in order to utilize redundant parts. The use of static reliability models for the dynamic case is equivalent to assuming unity probability of success of both actions; for this reason, very high reliabilities can be predicted as the number of spares is increased. It was recognized early in the studies of dynamic redundancy that imperfect detection and recovery may leave some spares unused.[33] The effect of such imperfections was formalized in the reliability model through the concept of "coverage", defined as the conditional probability of successful recovery, given that a fault has occurred.[31]

In general, the dynamic model[31,34,35] must represent the complete complexity of the proposed fault-tolerant system, including (1) differing failure rates for powered and unpowered modules; (2) number of spares of each module; (3) imperfect fault detection and recovery; (4) method of "hard core" protection; (5) existing intra-module fault tolerance; (6) extent and value of the faults; (7) duration and distribution of expected transient faults. Recent work has considered repairable systems[36] and models that include transient as well as permanent faults.[37,38] The principal objective of further study remains the development of models that integrate the characteristics of both hardware and software and consider both permanent and transient faults.

Analytic models of dynamically redundant systems are complex because of the number of different parameters that may be varied in the search for a balanced design. A very useful tool for reliability prediction is an interactive computer program which permits a ready variation of the important parameters of the redundant system for on-line design refinement. A pioneering effort in automated reliability modeling was the *REL* program, written in the APL language to predict system reliability for a given mission time when the system parameters have been specified.[31] Recent efforts to arrive at general and computationally efficient models have resulted in further APL programs.[38]

Experimental reliability prediction

Two approaches to experimental prediction of reliability are simulation and experimentation with a hardware prototype. While their use is more costly and time-consuming, the experimental methods are essential when the available analytic models do not adequately represent the complex structure of the system or the nature of the expected faults.

An accurate description of the system and characterization of the faults are the principal prerequisites when simulation is employed to derive the fault-tolerance estimates for the computer.[26] This approach has been extensively employed in reliability prediction for the redundant SATURN V guidance computer.[8,39] The use of hardware prototypes requires a large investment of effort in constructing the prototype, but avoids the inaccuracies which may occur in postulating the fault effects in a simulated model of the system. Two examples of use of hardware prototypes are: the switching system ESS No. 1 for which a catalog of fault symptoms was compiled by using a hardware model,[9] and the experimental fault-tolerant JPL-STAR computer.[11] In the JPL-STAR computer an electronic "black box" was used to inject faults of adjustable duration and extent at selected points in the hardware of the system. Another example of the experimental approach is the OAO processor[8] in which a component-redundant system was completely disassembled to determine the number of failed components after 3000 hours of operation.

A recent simulation and analysis system to analyze the behavior of faulty circuits is the LAMP (Logic Analyzer for Maintenance Planning) system.[40] In addition, LAMP also performs logic design verification, generates fault-detection tests, evaluates diagnostics, and produces trouble-location manuals. LAMP exemplifies the current trend toward multipurpose simulation systems in digital system design.

RELATIONSHIPS WITH THE SOFTWARE RELIABILITY PROBLEM

Discussion of the attributes of a fault-tolerant system included the goal that one of the two conditions should be satisfied with respect to design faults:

—the hardware and software should be free of design faults prior to the start of the computing process; or

—the system should contain complete provisions to detect and to circumvent the effects of hardware and software design faults during the computing process.

This means that the software of a fault-tolerant system must either be perfect (i.e., fault-free) or fault-tolerant in the same sense as the fault-tolerant hardware. The basic difference between the two is that operational faults in hardware occur after the start of the computing process, while design faults in software (and hardware, as well) are present at the start, but become disruptive only at a later time. However, software modifications and corrections of discovered design faults occasionally lead to new design faults and therefore the discoveries of software design faults may be expected throughout the useful life of any large software system, similar to the occurrence of operational hardware faults. This practically verified observation establishes the relationship between the methodologies for dealing with operational faults and design faults: the methods of protective redundancy that have proven successful in hardware fault-tolerance may be transferable to provide fault-tolerance of a software system as well.

An overview of the procedures currently used to attain software reliability shows that the "fault-intolerance" approach of perfecting the software prior to its regular use has been the accepted practice of improving software reliability. Three aspects of relevance of fault-tolerance can be identified:[41]

—the contribution of hardware fault-tolerant systems in assuring reliable computing;

—the common aspects of fault-tolerance that are equally applicable to hardware and software;

—the transfer of fault-tolerance techniques and experience from hardware to software, considering:

 —the applicability to software;

 —the potential advantages of software fault-tolerance;

 —the cost of its use, compared against the traditional fault-intolerance techniques.

The immediate advantage to a software system which results from the existence of a fault-tolerant hardware design is the protection of the software against disruptions caused by operational faults. In the case of a fault, the fault-tolerant features execute the corrective action in the hardware and restart the software, usually at a programmer-specified restart point,[14] although in some cases there is a single-instruction restart procedure which is transparent to the programmer.[27] The cost of utilizing the fault-tolerance features to achieve software protection consists of the programming constraints that must be observed to make automatic hardware initiated restarts of the programs possible. The advantages, in addition to the protection itself, also include the ability to distinguish, with a very high probability of success, whether a system crash was hardware-caused or not. Furthermore, a direct extension of the fault-tolerance techniques may be utilized to provide hardware-controlled protection of software and the data base against deliberate attempts to disrupt its operation and to access privileged information.

An area in which a common ground exists for hardware and software reliability efforts is the analytic modeling and quantitative prediction of system reliability.[31] The recent work on software reliability models[42,43] indicates the possibility of mutual reinforcement of research that would lead to the development of analytical models for the total system reliability, including both the hardware and software aspects. A second common area is design verification, in which the rapidly evolving techniques of program testing and proving have obvious applications to the problem of verifying hardware designs.

Finally, we consider the transfer to software of those protective redundancy techniques that have been successfully used in hardware system design. In the static approach, the same computation is carried out by two or more independently written programs.[41,44] The dynamic approach uses an analog of standby sparing with fault detection and switching of software modules.[45] While the cost aspect of both statically and dynamically fault-tolerant software remains to be explored, the continued use of 'pure' fault-intolerance for software reliability cannot be justified by tradition alone. It is hoped that the success of fault-tolerant hardware will stimulate further studies of the merits of fault-tolerance and redundancy in computer software.

REFERENCES

1. Carter, W. C. and W. G. Bouricius, "A Survey of Fault-Tolerant Computer Architecture and Its Evaluation," *Computer*, Vol. 4, No. 1, January-February, 1971, pp. 9-16.
2. Avižienis, A., "Architecture of Fault-Tolerant Computing Systems," *Digest of the 1975 International Symposium on Fault-Tolerant Computing*, Paris, France, June 1975, pp. 3-16.
3. Avižienis, A., "Design of Fault-Tolerant Computers," *AFIPS Conference Proceedings*, Vol. 31, 1967, pp. 733-743.
4. Avižienis, A., "The Methodology of Fault-Tolerant Computing," *Proceedings of the First USA-Japan Computer Conference*, Tokyo, Japan, October 1972, pp. 405-413.
5. Short, R. A., "The Attainment of Reliable Digital Systems Through the Use of Redundancy—A Survey," *IEEE Computer Group News*, Vol. 2, No. 2, March 1968, pp. 2-17.

6. Von Neumann, J., Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," in *Automata Studies*, C. E. Shannon and J. McCarthy, eds., Annals of Math. Studies No. 34, Princeton, University Press, 1956, pp. 43-98.
7. Moore, E. F. and C. E. Shannon, "Reliable Circuits Using Less Reliable Relays," *Journal of the Franklin Institute*, Vol. 262, Nos. 9 and 10, September, October 1956, pp. 191-208 and 281-297.
8. Cooper, A. E. and W. T. Chow, "Development of On-Board Space Computer Systems," *IBM Journal of Research and Development*, Vol. 20, No. 1, January 1976, pp. 5-19.
9. Downing, R. W., J. S. Nowak and L. S. Tuomenoksa, "No. 1 ESS Maintenance Plan," *The Bell System Technical Journal*, Vol. 43, No. 5, Part 1, September 1964, pp. 1961-2019.
10. Beuscher, H. J. et al., "Administration and Maintenance Plan of No. 2 ESS," *The Bell System Technical Journal*, Vol. 48, October 1969, pp. 2765-2815.
11. Avižienis, A., G. C. Gilley, F. P. Mathur, D. A. Rennels, J. A. Rohr and D. K. Rubin, "The STAR (*S*elf-*T*esting-*A*nd *R*eparing) Computer: An Investigation of the Theory and Practice of Fault-Tolerant Computer Design," *IEEE Transactions on Computers*, Vol C-20, No. 11, November 1971, pp. 1312-1321.
12. Carter, W. C., D. C. Jessep, P. R. Schneider, A. B. Wadia and W. G. Bouricius, "Logic Design for Dynamic and Interactive Recovery," *IEEE Transactions on Computers*, Vol. C-20, No. 1, November 1971, pp. 1300-1305.
13. "An Application-Oriented Multiprocessing System," *IBM Systems Journal*, Vol. 6, No. 2, 1967.
14. Rohr, J. A., "STAREX Self-Repair Routines: Software Recovery in the JPL-STAR Computer," *Digest of the 1973 International Symposium on Fault-Tolerant Computing*, Palo Alto, CA, June 1973, pp. 11-16.
15. Carter, W. C. and P. R. Schneider, "Design of Dynamically Checked Computers," *Information Processing 68* (Proceedings of IFIP Congress 1968), pp. 878-883.
16. Conn, R. B., N. A. Alexandridis and A. Avižienis, "Design of a Fault-Tolerant Modular Computer with Dynamic Redundancy," *AFIPS Conference Proceedings*, Vol. 41, (Fall JCC 1972), pp. 1057-1067.
17. Gilley, G. C., "A Fault-Tolerant Spacecraft," *Digest of 1972 International Symposium on Fault-Tolerant Computing*, June 1972, pp. 105-109.
18. Maison, F. P., "The MECRA: A Self-Repairable Computer for Highly Reliable Process," *IEEE Transactions on Computers*, Vol. C-20, No. 11, November 1971, pp. 1382-1393.
19. Everett, R. R., C. A. Zraket and H. D. Benington, "SAGE—A Data-Processing System for Air Defense," *Proceedings of the Eastern Joint Computer Conference*, Washington, D.C., December 1957, pp. 148-155.
20. Ornstein, S. M., et al. "Pluribus—A Reliable Multiprocessor," *AFIPS Conference Proceedings*, Vol. 44, 1975 NCC, pp. 551-559.
21. Blaauw, G. A., "The Structure of SYSTEM/360: Part V—Multisystem Organization," *IBM System Journal*, Vol. 3, No. 2, 1964, pp. 181-195.
22. *A Guide to the IBM System/360 Model 145*, IBM Corporation, Technical Publications Department, White Plains, New York, Third Edition, August 1972.
23. Corbato, F. J., J H. Saltzer and C. T. Clingen, "Multics—The First Seven Years," *AFIPS Conference Proceedings*, Vol. 40 (Spring JCC 1972), pp. 571-583.
24. Borgerson, B. R., "Dynamic Confirmation of System Integrity," *AFIPS Conference Proceedings*, Vol. 41, Part 1, 1972, pp. 89-96.
25. Wensley, J. H., "SIFT—Software Implemented Fault Tolerance," *AFIPS Conference Proceedings*, Vol. 41, Part 1, 1972, pp. 243-254.
26. Levy, H. O. and R. B. Conn, "A Simulation Program for Reliability Prediction of Fault Tolerant Systems," *Digest of the Fifth International Symposium on Fault-Tolerant Computing*, Paris, France, June 1975, pp. 104-109.
27. Hopkins, A. L., Jr. and T. B. Smith III, "The Architectural Elements of A Symmetric Fault-Tolerant Multiprocessor," *IEEE Transactions on Computers*, Vol. C-24, No. 5, May 1975, pp. 498-505.
28. Tang, D. T. and R. T. Chien, "Coding for Error Control," *IBM Systems Journal*, Vol. 8, No. 1, 1969, pp. 48-86.
29. Parhami, B. and A. Avižienis, "A Study of Fault-Tolerance Techniques for Associative Processors," *AFIPS Conference Proceedings*, Vol. 43, 1974, pp. 643-652.
30. Avižienis, A., "Arithmetic Error Codes: Cost and Effectiveness Studies for Application In Digital System Design," *IEEE Transactions on Computers*, Vol. C-20, No. 11, November 1971, pp. 1322-1331.
31. Bouricius, W. G., W. C. Carter and P. R. Schneider, "Reliability Modeling Techniques for Self-Repairing Computer Systems," *Proceedings of the 24th National Conference of ACM*, 1969, pp. 295-383.
32. Barlow, R. W. and F. Proschan, *Mathematical Theory of Reliability*, Wiley and Sons, 1965.
33. Griesmer, J. E., R. E. Miller, J. P. Roth, "The Design of Digital Circuits to Eliminate Catastrophic Failures," *Redundancy Techniques for Computing Systems*, Spartan Press, Inc., Washington, D.C., 1962, pp. 328-348.
34. Bricker, J. L., "A Unified Method for Analyzing Mission Reliability for Fault-Tolerant Computer Systems," *IEEE Transactions on Reliability*, Vol. R-22, No. 2, June 1973, pp. 72-77.
35. Ng, Y. W. and A. Avižienis, "A Unifying Reliability Model for Closed Fault-Tolerant Systems," *Digest of the Fifth International Symposium on Fault-Tolerant Computing*, Paris, France, June 1975, p. 224; full text to appear in IEEE Transactions on Computers.
36. Arnold, T. F., "The Concept of Coverage and Its Effect on the Reliability Model of a Repairable System," *IEEE Transactions on Computers*, Vol. C-22, No. 3, March 1973, pp. 251-254.
37. Merryman, P. M. and A. Avižienis, "Modeling Transient Faults in TMR Computer Systems," *Proceedings 1975 Annual Reliability and Maintainability Symposium*, Washington, D.C., January 1975, pp. 333-339.
38. Ng, Y. W., "Modeling and Analysis of Fault-Tolerant Computers," Ph.D. Dissertation, UCLA, Computer Science Department, University of California, Los Angeles, 1976.
39. Hardie, F. H. and R. S. Suhocki, "Design and Use of Fault Simulation for Saturn Computer Design," *IEEE Transactions on Computers*, Vol. EC-16, August 1967, pp. 412-429.
40. Chang, H. Y., G. W. Smith, Jr. and R. B. Walford, "LAMP: System Description," *The Bell System Technical Journal*, Vol. 53, No. 8, October 1974, pp. 1431-1449.
41. Avižienis, A., "Fault-Tolerance and Fault-Intolerance: Complementary Approaches to Reliable Computing," *Proceedings of the 1975 International Conference on Reliable Software*, Los Angeles, California April 1975, pp. 458-464.
42. Shooman, M. L., "Operational Testing and Software Reliability Estimation During Program Development," *Proceedings of the 1973 IEEE Symposium on Computer Software Reliability, New York City*, 1973, pp. 51-56.
43. Moranda, P. B., "Prediction of Software Reliability during Debugging," *Proceedings of the 1975 Annual Reliability and Maintainability Symposium*, January 1975, pp. 327-332.
44. Elmendorf, W. R., "Fault-Tolerant Programming," *Digest of the 1972 International Symposium on Fault-Tolerant Computing*, IEEE Computer Society, 1972, pp. 79-83.
45. Randell, B., "System Structure for Software Fault Tolerance," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 2, June 1975, pp. 220-232.

Failure-tolerant parallel programming and its supporting system architecture

by K. H. KIM
University of Southern California
Los Angeles, California

and

by C. V. RAMAMOORTHY
University of California
Berkeley, California

ABSTRACT

The state-of-art in software validation as well as the continuing growth of the size and complexity of software subsystems, makes extra costs paid for software error tolerance more than justified. A program in which software redundancy is incorporated i.e. a program in which procedures for run-time validation and recovery are explicitly specified, is generally called a failure-tolerant program. One problem in failure-tolerant programming, which could be particularly serious in real-time computing environments, is the program execution time increased due to incorporation of validation and recovery procedures. This paper introduces an approach to the solution, called the failure-tolerant parallel programming. The essence of this approach is to maximally overlap main-stream computation with redundant computation oriented for validation and recovery. Subsequently, a model system architecture tailored for efficient execution of failure-tolerant parallel programs is described. It is of highly general and modular nature and contains a novel memory subsystem named the duplex memory. Directions of further researches on program structuring and expansion of the model architecture are also indicated.

INTRODUCTION

Computing system reliability is a function of both *hardware reliability* and *software reliability*. Hardware failures occur due to physical component *faults* (i.e., material characteristics) or *design errors*. The former source has been dominating the latter in significance. Recent advances in hardware component technology have substantially reduced the occurrences of hardware faults, thus greatly improving hardware reliability. On the other hand, all software failures are due to design errors. As the size and complexity of software subsystems grows steadily, software reliability has become a very serious problem and an increasingly important factor determining the overall system reliability.

Complete validation, which assures the absolute correctness of a program through verification of its complete behavioral characteristics, still remains to be infeasible with sizable programs.[2,12] On the other hand, more popular and pragmatic approaches aiming at *partial validation* with high cost-effectiveness via testing cannot, by their nature, insure the absence of errors in the program.[10,13,16,20,21] The apparent consequence is the current practice in which errors remain to exist in large programs put into operation. It is also this practice that makes *software error tolerance* an important objective besides complete removal of software errors at the design stage.

The concept of *failure-tolerant computing* i.e., reliable computing despite the presence of system component failures, was born in the very early days of electronic computing.[18,26] Since then, *hardware fault tolerance* has been a main subject of extensive investigation.[1,3,7,25] *Redundancy* is a fundamental vehicle in realizing failure-tolerant computing. A majority of previous studies have been centered around the use of hardware redundancy and in contrast, very little studies were made on the use of software redundancy. A restricted amount of software redundancy has been exploited in the form of *rollback and recovery* defined as follows. Let *state vector* refer to a snapshot of the contents of all the variables of the program in execution. Rollback and recovery is a technique of depositing state vectors at several stages in the middle of program execution and in case of a system failure, resetting the system state by using an old state vector and restarting the execution from that stage. However, the way failure detection, state vector saving and recovery operations are designed and specified has been mostly ad hoc and heuristic. It was only in recent years that studies were made on systematic and cost-effective implementation of a rollback and recovery scheme.[1,5,6,15,19,22,24]

In case of hardware faults, rollback (possibly combined with system reconfiguration using redundant hardware components) and re-execution with the same program will suffice to get over the situation. However, such an approach does not help in case of software failures. From the very nature of software errors, software error tolerance requires more extensive exploitation of redundancy, particularly software redundancy which is essentially a design redundancy. The method of structuring programs in which software redundancy is explicitly incorporated, is generally called *failure-tolerant programming.* It was in recent years that software error tolerance became a subject of serious studies and research was initiated toward the development of structured failure-tolerant programming techniques.[8,11,14,17,22,27]

In the next section, a brief overview of those recent significant contributions is given. Then some desirable directions of extending the state-of-art in failure-tolerant programming, which, we believe, are significant in real-time computing environments, are pointed out. The following section introduces a new approach to failure-tolerant programming (termed failure-tolerant parallel programming) devised to be a desirable extension of the state-of-art, and discusses the requirements on the system architecture oriented for efficient execution of failure-tolerant parallel programs. The following section describes an architecture developed to satisfy the requirements discussed in the preceding section. Finally, areas of extension and further research are discussed and then this paper is concluded.

BACKGROUND

Recent research on failure-tolerant programming and software error tolerance made significant contributions in the following aspects:

First, the notion of a *failure-tolerant program* was solidified. A failure-tolerant program is essentially a self-checking and recovering program. More specifically, a failure-tolerant program contains specifications of the procedures of validating intermediate results at various stages during execution and recovering when an abnormal condition is detected as a result of the check. Thus it consists of two types of program-segments: (1) *object segments* specifying application-oriented computations, and (2) *validation and recovery* (*VR-*) *segments,* each associated with a certain object segment and specifying the procedures of validating the results produced by the associated object segment and recovering in case of incorrect results. Within a failure-tolerant program, powerful facilities for validation and recovery can be incorporated in a systematic manner to any desirable extent. Here "recovery" implies not just the repetition of the execution with the same object segment (which may have failed the validation-test due to the hardware faults or the errors contained in it) but rather the provision of a set of "alternate" object segments and trials with one after another until a certain alternate object segment passes the validation test. If all the alternatives fail, then either the program cannot be successfully completed or a more global recovery action is incurred, provided the failed object segments are nested in another object segment and the latter is associated with a VR-segment.

Importance of good structure in failure-tolerant programs is evident, since structuring a failure-tolerant program by introducing VR-segments into a conventional program containing only object segments is accompanied by an increase in program size and complexity. Recognizing this importance, Randell's group at the University of Newcastle upon Tyne, England developed an experimental scheme called *recovery block structuring* by which validation and recovery functions can be embedded, in a well-structured form, inside each block in programs written in block-structured languages like ALGOL.[14,22] To give some flavor to this structuring scheme, the structure of the *recovery block* (i.e., the failure-tolerant block) is depicted in Figure 1.

In the diagram, double vertical lines define the bodies (i.e. scopes) of recovery blocks, while single vertical lines define the bodies of primary or alternate object blocks. The *primary object block* corresponds exactly to the block of the equivalent conventional

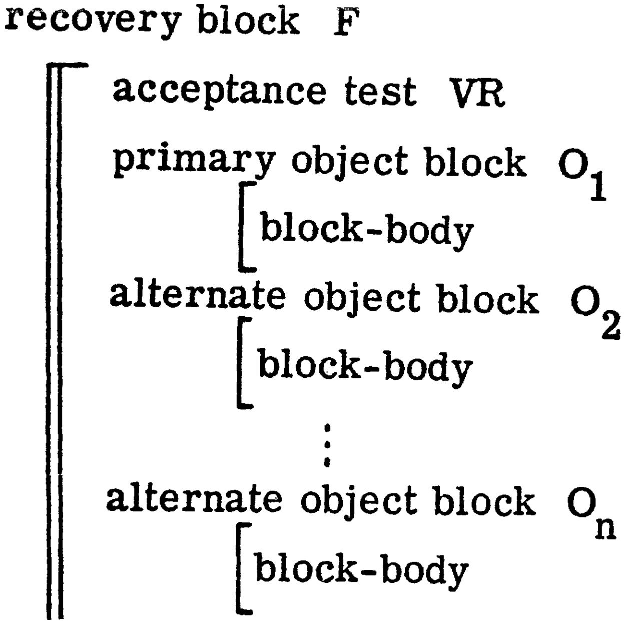


Figure 1—A structural model of the recovery block developed in References 14 and 22

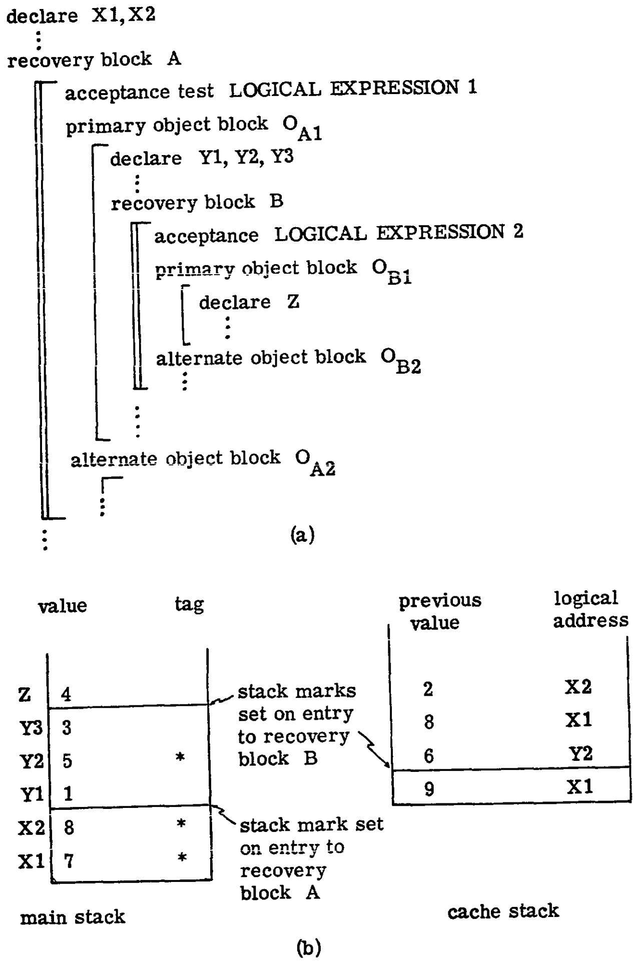
(ALGOL-like) program and is a kind of an object segment. The *acceptance test* is executed on exit from an object block to confirm that the object block has performed acceptably. If confirmed, the control exits from the recovery block. Thus the acceptance test is a kind of a VR-segment. If the result produced by an object block is determined to be unacceptable, the next *alternate object block* is entered and is required to perform the objective operation in a different way or to perform some alternate action acceptable to the program as a whole. The acceptance test is then repeated.

The following aspects of recovery block structuring are rather fundamental and may be found, possibly in different formats, in any structured approach to the failure-tolerant programming.

(1) The primary or alternate object blocks can contain, nested within themselves, further recovery blocks.
(2) When an alternate object block needs to be entered after the result of the preceding object block fails the acceptance test, the system state must be restored to the one current just before entry to the primary object block.
(3) Execution of the acceptance test upon exit from an object block generally requires the reference to both the original values and the modified values of the variables non-local to the object block.
(4) It is not necessary that every block in a block-structured failure-tolerant program be a recovery block.

Second, a technical basis was established for reducing the overhead involved in saving a state vector on entry to each object segment and resetting the system state by using a saved state vector during recovery. The overhead exists in two forms. One is the processor time spent for those activities and the other is the store space occupied by saved state vectors. A useful property which can be advantageously exploited for overhead reduction is that the variables local to the object segment are irrelevant to the recovery and in many cases, only a few of the non-local variables are modified by the object segment.

Based on this, Randell's group developed a scheme for state vector saving and system state resetting, called a *recursive cache mechanism,* to support execution of programs structured by the scheme of recovery block structuring.[14] The essence of this scheme is to save the original value of each non-local variable together with its name (i.e., its logical address) right before the variable is modified for the first time in a new object block. Thus state vectors are saved in compact forms. It is apparently necessary to detect, at run-time, whether an assignment to a non-local variable is the first to have been made to that variable within the current block. This capability is provided by the *flag* attached to each non-local variable. Again, to give some flavor to this mechanism, an example of the recursive cache is shown in Figure 2.



```
declare X1,X2
  :
recovery block A
  acceptance test LOGICAL EXPRESSION 1
  primary object block O_A1
    declare Y1, Y2, Y3
      :
    recovery block B
      acceptance LOGICAL EXPRESSION 2
      primary object block O_B1
        declare Z
          :
      alternate object block O_B2
          :
        :
  alternate object block O_A2
      :
  :
```

(a)

(b)

Figure 2—(a) A program structured by the recovery block structuring (b) A snapshot of the recursive cache during execution of (a)

Figure 2(a) shows a failure-tolerant program structured by the recovery block structuring scheme. Figure 2(b) shows a snapshot of the recursive cache taken when primary object block O_{B1} is in the middle of its execution. There are two stacks, the *main stack* and the *cache stack*. The cache stack is also divided into regions, one for each nested recovery block in "active" state. The top region of the cache stack in Figure 2(b) contains previous values of non-local variables together with their names i.e., Y2, X1, X2, which have been modified by execution of the current object block O_{B1}. The flags attached to those non-local variables in the main stack are set accordingly. Similarly, the bottom region of the cache stack contains the previous value of non-local variable X1 which had been modified by execution of object block O_{A1} before O_{B1} was entered. If the result produced by execution of O_{B1} fails the acceptance test (LOGICAL EXPRESSION 2), then the top region of the cache stack can be used to reset the content of the main stack to the one current before entry to recovery block B. If it passes the test, the top region is merged into the bottom region of the cache stack so that the result will contain previous values of those variables which are non-local to object block O_{A1} and have been modified since O_{A1} was entered. Thus the result will be a single region containing (9, X1) and (2, X2). Flags in the main stack are also adjusted such that only flags of X1 and X2 be set. Therefore, the combination of the main and cache stacks contain information on the basis of which several old state vectors can be reconstructed.

This is perhaps an oversimplified account of recent developments. Yet it is intended to provide all the essential backgrounds for clarifying main departures of our works presented in the rest of this paper. For more information on the schemes described in this section, readers are of course referred to their original reports.[14,22]

DESIRABLE EXTENSIONS OF THE STATE-OF-ART IN FAILURE-TOLERANT PROGRAMMING

On the basis of recent works in failure-tolerant programming, particularly those introduced in the preceding section, various extensions can be clearly envisioned. Among many desirable extensions, the following ones are considered to be of great significance.

First, one problem in failure-tolerant programming, which could be particularly serious in real-time computing environments, is the program execution time increase due to incorporation of VR-segments. In most of the previous approaches including the recovery block structuring and recursive cache schemes introduced in the preceding section, validation and saving of state vectors fully contribute to the increase of the program execution time. Consequently, when any nontrivial validation is employed or large numbers of non-local variables are modified during execution of object segments, the program execution time, even in the case of normal failure-free operation, could very well exceed the tolerable limit in real-time applications. It is indeed expected that the VR-segment will be frequently a quite complex program-segment. Even in the recovery block structuring scheme in which only a restricted form of a VR-segment i.e. a logical expression is allowed for the sake of reducing errorproneness of the VR-segment, a provision has been made to allow procedure calls within acceptance tests (i.e., logical expressions).[14]

Second, the rationale underlying the restriction of the acceptance test to a logical expression is considered a perfectly legitimate one. Yet the logical expression is considered an excessively restrictive form of a VR-segment in many environments. For instance, it may be desirable to explicitly specify in the VR-segment which alternate object segment, among multiple alternates, is to be tried next in each case of recovery, rather than always letting the system select the next alternate object segment randomly or in the order alternates are located in the program text. It may also be desirable to immediately revert to a global recovery if a certain erroneous condition is detected by execution of a VR-segment, instead of retrying with an alternate. That is, allowance of more flexible structures in failure-tolerant programs may be desirable. Furthermore, our approach toward the first desirable extension mentioned above i.e., execution of VR-segments with minimal increase in the overall program execution time, favors more flexibility in structuring failure-tolerant programs. This will become evident in the next section.

CONCEPT OF FAILURE-TOLERANT PARALLEL PROGRAMMING AND REQUIREMENTS ON THE SUPPORTING SYSTEM ARCHITECTURE

Our main concern in this paper is with the first desirable extension mentioned in the preceding section, that is, incorporation of VR-segments with minimal increase of program execution time.

The fundamental approach we have adopted is to maximally overlap execution of object segments with execution of VR-segments. Since the VR-segment specifies manipulation on the results produced by its associated object segment, dependency of the former for its initiation on the completion of the latter is inherent. However, it is possible to execute the VR-segment associated with an object segment concurrently with the successor object segment(s). Figure 3 illustrates this concept. There VR-segment VR_2 can be initiated only after completion of the correspondent object segment O_2 and VR-segment VR_1, but it may be executed concurrently with object segments O_3, O_4, etc. In an ideal situation where execution of VR-segments is fully overlapped with execution of object segments, the amount of increase in program execution time will be the time required for execution of the last VR-segment (e.g., VR_n in Figure 3) since it is the only VR-segment which cannot be executed in overlap with object segments.

A failure-tolerant program in which computational parallelism, especially parallelism between application-oriented computations and redundant computations for validation and recovery, is explicitly indicated, is called a *failure-tolerant parallel program*. That is, the main type of parallelism which characterizes a failure-tolerant parallel program is the one existent between object segments and VR-segments.

This approach requires a new method of structuring a failure-tolerant program. The major departure of a newly required structuring method from the previously developed ones is in specification of the control structure among program-segments. In addition to inherent dependency of VR-segments on their correspondent object segments, dependency of object segments on VR-segments may also be specified in a failure-tolerant

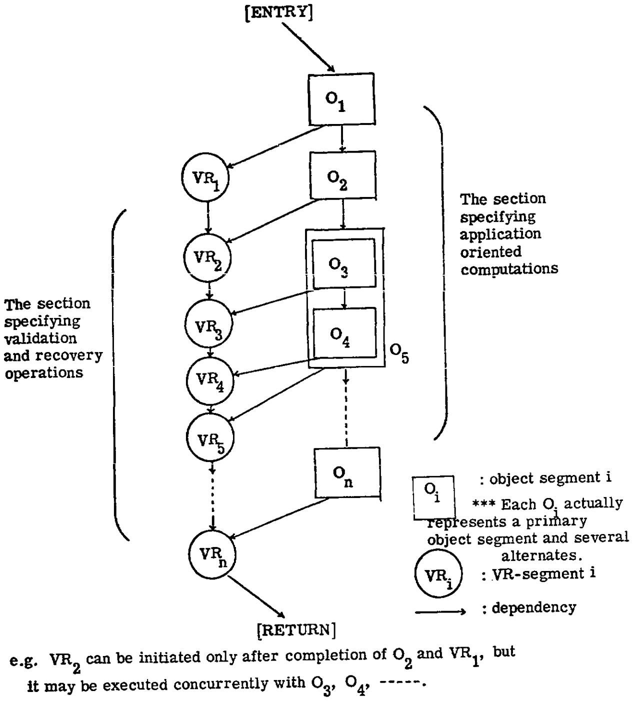

[ENTRY]
O1
O2
O3
O4
O5
On
VR1
VR2
VR3
VR4
VR5
VRn
The section specifying application oriented computations
The section specifying validation and recovery operations
Oi : object segment i
*** Each Oi actually represents a primary object segment and several alternates.
VRi : VR-segment i
→ : dependency
[RETURN]
e.g. VR2 can be initiated only after completion of O2 and VR1, but it may be executed concurrently with O3, O4, -----.

Figure 3—A simple example of a failure-tolerant parallel program

parallel program. An example of such a situation is when a certain object segment specifies a critical operation such as ordering an emergency action, erasing a secret file, etc. In such a case it is desirable to suspend the execution of the object segment until all the VR-segments corresponding to its predecessor object segments have been verified. Thus critical operations can be controlled to occur reliably. This clearly favors, if not requires, the second desirable extension discussed in the preceding section i.e., more flexibility in structuring VR-segments and overall failure-tolerant programs. Yet such flexibility can be obtained without sacrificing most desirable structuring principles (or strategies) underlying the developed structuring schemes, including the recovery block structuring scheme. This aspect will be discussed later.

On the other hand, the failure-tolerant parallel programming imposes the following requirements on the supporting system architecture i.e., the architecture of a system capable of efficient execution of failure-tolerant parallel programs.

First and the most obvious of all, the system must contain at least two processors, one for execution of object segments, called the *object processor* and the other for execution of VR-segments, called the *VR-processor*.

Second, the state vector at the completion of an object segment is an input data not only to the successor object segment but also to the associated VR-segment. In the rest of this paper, a state vector refers to a snapshot of all the variables appearing in object segments, not including local variables defined in VR-segments, taken at one moment. Furthermore, the VR-segment requires the input state vector undestroyed until it no longer needs to examine it, while the successor object segment, by nature, continuously changes it into the up-to-date one. It is thus necessary to create a "copy" of the state vector current at the end of the execution of an object segment for exclusive use by the VR-processor executing the associated VR-segment. Here the process of creating a copy must cause no or little delay in executing object segments. The store space containing copies of state vectors must also be minimized. If either execution time or store space exceeds the tolerable limit, the objective of failure-tolerant parallel programming is defeated. This requires a new store management scheme substantially different from the previously developed ones. This point will become clearer when we propose one suitable scheme in the next section.

Third, if execution of VR-segments lags much behind execution of object segments, there will be accumulated a large number of unprocessed copies of state vectors. Thus when all the available store space runs out, the object processor must be suspended until the VR-processor catches up. In order to avoid this undesirable situation, execution of VR-segments must be speedy.

A MODEL ARCHITECTURE SUPPORTING FAILURE-TOLERANT PARALLEL PROGRAMMING

In this section we describe a system architecture oriented for efficient execution of failure-tolerant parallel programs. We call it a model architecture since it is of highly general nature and thus is specified at an abstract level. Yet it is expected that elaboration of the architecture into a specific working system will encounter no new logical problems of fundamental nature.

The store management scheme

Potential power of failure-tolerant parallel programs cannot be realized without accompanying the additional cost of the supporting system architecture. The additional cost is paid in the forms of both processor redundancy and store redundancy. Since the object processor runs concurrently with the VR-processor, memory conflicts must be carefully avoided. This rules out the feasibility of having a single state vector or its portion shared by both processors. Thus each processor owns a region of the store during execution of a failure-tolerant parallel program.

The region of the store used by the object processor is called the *main working store*, while the region of the store used by the VR-processor is called the *VR-store.* The VR-store contains copies of state vectors including the up-to-date one plus possibly more than one old one. This, of course, does not mean that the VR-store contains complete duplicates of several state vectors. Let S_i, S_{i+1}, ------, S_{i+j} denote a (chronological) sequence of state vectors that can be reconstructed from the content of the VR-store. Then the VR-store actually contains one complete copy of S_i (i.e., the oldest reconstructable one) and only the differences between pairs of adjacent state vectors in the sequence i.e. $S_{i+1}-S_i$, $S_{i+2}-S_{i+1}$, -------, $S_{i+j}-S_{i+j-1}$.

More specifically, as the object processor executes each object segment, it produces the *execution image* which consists of the values of variables assigned during execution of that object segment. If a variable is assigned several times during execution of the object segment, only the latest assigned value is contained in the execution image. The execution image produced on completion of an object segment represents the difference between the state vector current right before entry to the object segment and the up-to-date state vector (i.e. the one current on completion of the object segment). Each execution image is stored in a segment of the VR-store called a *VR-store-segment.* Each execution image is examined by the VR-processor to determine the acceptability of the result produced by execution of the object segment.

The execution image of an object segment consists of values of both local variables and non-local variables. Thus each VR-store-segment consists of two sections, one for local variables and the other for non-local variables. On entry to each object segment, memory space is allocated for the segment of the main working store containing the set of local variables defined within the object segment. At that time, the same size of memory space is also allocated for the section of the VR-store-segment containing (a copy of) the set of local variables. However, store space for non-local variables is not entirely duplicated. Instead, the section of the execution image containing non-local variables is written in the form of a table in which each entry consists of the logical address and the new value of a non-local variable. The idea is to take advantage of the useful property that in many cases, only a few of the non-local variables are modified by an object segment, while the total number of non-local variables defined may be very large. Thus the table representation leads to a highly compact form of the non-local variable section of the execution image.

As a simple example, consider a block-structured program augmented with VR-segments in Figure 4(a). Figure 4(b) shows snapshots of the main working store i.e. the stack used during execution of object segments in the program. In Figure 4(c) VR-store-segment 1 (or 2, 3, 4, 5) is used to contain the execution image of object segment O_1 (or O_2, O_3, O_4, O_5). Each VR-store-segment except VR-store-segment 1 consists of two sections, one for local variables and the other for a table holding newly assigned values of non-local variables and their logical addresses. Each VR-store-segment is created on entry of the object processor to the correspondent object segment.

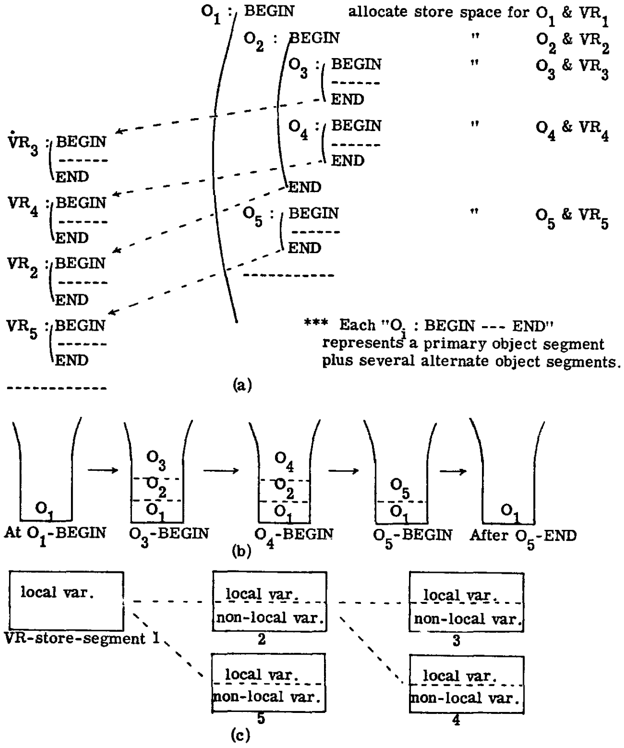


Figure 4—(a) A failure-tolerant parallel program (b) Snapshots of the mail working store during execution (c) A snapshot of the VR-store during execution

At the beginning of VR-segment VR_3, VR-store-segment 3 contains the execution image of object segment O_3 and the object processor has probably entered into O_4. The execution image in VR-store-segment 3 is examined by execution of VR_3. When it has been verified or judged to be acceptable, the local variable section is discarded and the non-local variable section is merged into VR-store-segment 2. If two different values of the same variable were contained in both VR-store-segments 3 and 2, the value in VR-store-segment 2 is the older one and replaced by the value in VR-store-segment 3. At the beginning of VR_4, VR-store-segment 3 does not exist. Then it is no longer possible to reconstruct the state vector which was current right before entry to O_3. There will be no need to reconstruct that state vector since VR_3 has been successfully completed.

Similarly, upon successful completion of VR_2, the values of non-local variables in VR-store-segment 2 are absorbed into VR-store-segment 1. At the beginning of VR_5, VR-store-segments 2, 3 and 4 do not exist. Then the state vector which was current at the initiation of O_2 can no longer be reconstructed.

Here one subtle problem is noteworthy. Consider a simple program in Figure 5(a). Since there are two object segments (thus two VR-segments), there are two VR-store-segments (1 and 3). The body of object segment 1 consists of the upper section, O_3 and the lower section. During execution of the upper section, the execution image will be stored in VR-store-segment 1. At the completion of O_3, VR-store-segment 3 contains the execution image and the VR-processor may start examining it. Then the object processor executes the lower section of O_1 and it stores the execution image into VR-store-segment 1 which already contains the execution image of the upper section. Some values in VR-store-segment 1 are now the ones assigned by execution of the upper section, while others are the ones assigned by execution of the lower section. When the VR-processor has successfully completed VR_3 and needs to merge the verified execution image into VR-store-segment 1, it is not possible to tell for each variable whether the value in VR-store-segment 1 is the older one than the one in VR-store-segment 3.

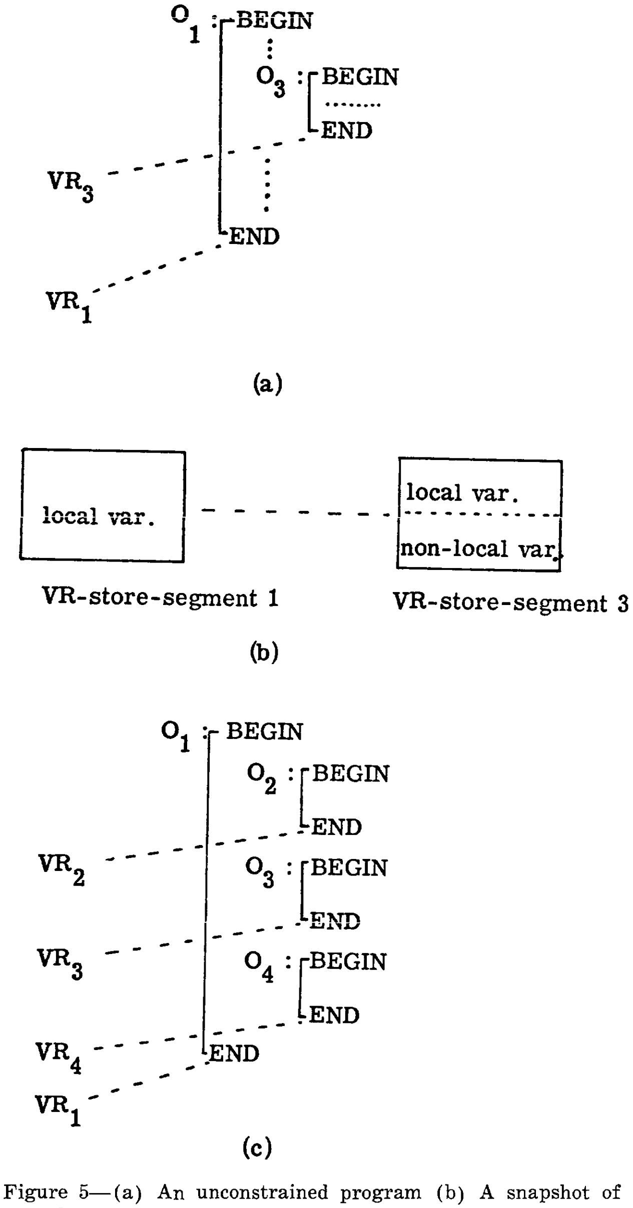


Figure 5—(a) An unconstrained program (b) A snapshot of the VR-store (c) An equivalent program satisfying the constraint

This problem can be resolved by imposing an additional constraint on program structuring. That is, the upper section and the lower section of O_1 in Figure 5(a) must be changed into O_2 and O_4, respectively, nested within O_1. This results in Figure 5(c). Then there will be additional VR-store-segments (2 and 4) created during execution, and the above problem disappears. Therefore, the constraint is that either each object segment contains no other object segments nested in it or its entire body must be composed of other object segments nested in it. This constraint is incorporated only for secure allocation of the VR-store. Thus the programmer is not required to prepare new VR-segments VR_2 and VR_4.

If there are no explicit VR-segments associated with some object segments, the system will insert dummy VR-segments. Or a variation of the above solution is to make the system responsible for restructuring unconstrained programs (e.g., Figure 5(a)) into the ones satisfying the constraint (e.g., Figure 5(c)), rather than imposing the constraint on the programmer. In any case, the solution does not appear to be a costly one.

The model system structure

The store management scheme described above will work (almost) perfectly if execution images can be created in the VR-store without causing any delay in execution of object segments. As far as the local variable section of the VR-store is concerned, this condition can be met without employing unconventional hardware components. Since the same size of the store-segment for local variables is allocated in both the main working store and the VR-store, the only requirement is to set proper base addresses for both store-segments and, for each assignment, write the same value in two locations of the same relative address, one in the main working store and the other in the VR-store.

However, the situation is different in creating the non-local variable section of the execution image. As described before, the non-local variable section is written in the form of a table. Thus whenever a non-local variable is assigned a new value during execution of an object segment, the value is written into the correspondent location in the main working store and at the same time, the value together with the logical address is written into an entry of a table in the VR-store.

If the assignment to a non-local variable is the first to have been made to that variable within the current object segment, then the new value and the logical address of that variable are written into the next available (empty) entry of the table in the VR-store. If it is not the first, there are two choices. One is to write in the same manner as above i.e. write into a new entry of the table. The other is to locate the entry of the table containing the previous value and the logical address of the non-local variable and then replace the previous value with the newly assigned one. After all, only the latest assigned values of variables need to be contained in the execution image. The former approach leads to the larger table-size than the one resulting from the latter approach. In addition, when or before the VR-processor executes the VR-segment to validate the table, it must discard the entry (of the table) containing the older value of each variable for which there is another entry containing the later assigned value. The latter approach leads to the compact table but it requires a special hardware support in order not to degrade the performance of the object processor. The special hardware requirement can be met by incorporation of the *content-addressable* (*i.e., associative*) *memory module* whose access time closely matches the one of the location-addressed memory module used in the system. In addition, incorporation of such a memory module significantly enhances the performance of the VR-processor in executing VR-segments. In view of the decreasing trend of the hardware cost and the performance advantage, the latter approach is considered favorable.

It is also necessary to attach a tag to the logical address of each variable which indicates whether the variable is a local one or a non-local one.

The structure of the model system in which all the above decisions are reflected, is depicted in Figure 6. The model system contains multiple central processors (CP's) and the memory subsystem named the *duplex memory*.

Each CP may function as an object processor, a VR-processor, a supervisory processor or a spare at one time. Employment of general purpose CP's is motivated mainly by the consideration of the flexibility in system reconfiguration. Yet this is not an absolute necessity and can be compromised for employment of processors fixed for a specific function if other factors such as cost and performance dictate so.

The duplex memory contains two types of memory modules, location-addressed memory modules and content-addressable memory modules. Location-addressed memory modules are further divided into two sets. One set of modules provides the main working store for the object processor. The other set, together with the set of content-addressable memory modules, provides the VR-store. Thus the local variable section of each execution image is contained in location-addressed memory modules, while the non-local variable section is contained in content-addressable memory modules. That is, whenever the object processor issues a "write" command, the value is written into two locations simultaneously, one in the main working store and the other in the VR-store. The latter location is in a location-addressed memory module if the tag attached to its logical address indicates "local" or in a content-addressable memory module if the tag indicates "non-local."

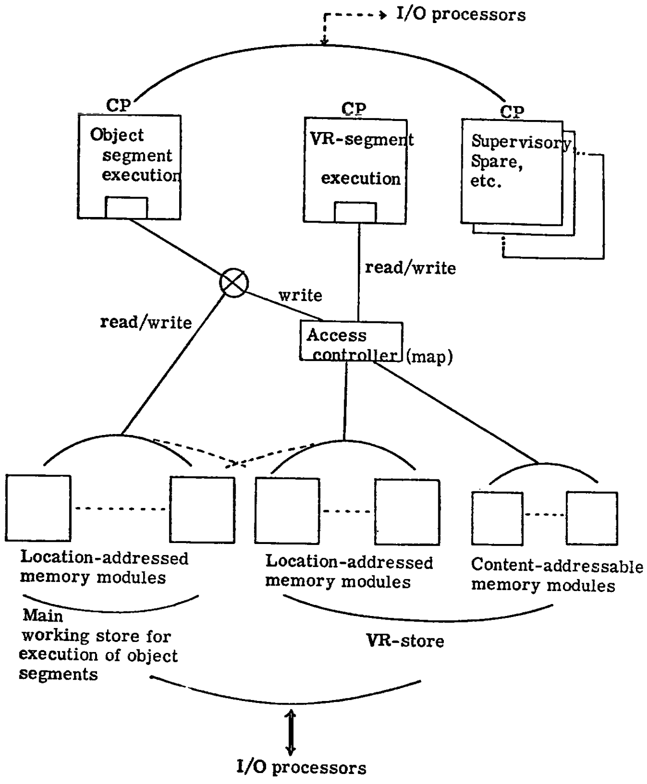


Figure 6—A system architecture based on multiple CP's and the duplex memory

The VR-processor never accesses the main working store except during recovery. The object processor never reads from the VR-store. It is a natural property of this duplex memory that the partition of location-addressed memory modules consisting of two disjoint sets, (one providing the main working store and the other providing the local variable section of the VR-store) may change dynamically.

There is another important reason for employing content-addressable memory modules. Execution of a VR-segment generally involves not only examination of the correspondent execution image but also references to other *ancestor* VR-store-segments i.e., ones created prior to the VR-store-segment containing the correspondent execution image. For instance, when the VR-processor executes VR-segment VR_4, in Figure 4, it examines the content of the correspondent VR-store-segment 4 and it may access ancestor VR-store-segments 1 and 2. References to ancestor VR-store-segments are for obtaining the previous values of non-local variables (i.e., variables non-local to the object segment associated with the VR-segment currently in execution).

It is frequently required for each non-local variable to obtain the latest assigned value among all of its values contained in ancestor VR-store-segments. The desired value may exist in the local variable section of an ancestor VR-store-segment. In this case, the variable is defined within the object segment which created the ancestor VR-store-segment. The desired location (which is in a location-addressed memory module) is exactly the one to which the logical address of the variable is mapped, and thus it can be directly accessed.

On the other hand, the desired value may exist in the non-local variable section of an ancestor VR-store-segment. For instance, assume in Figure 4 that variable Z is defined in object segment O_1, initialized with "100," assigned "200" by execution of O_3 and assigned "300" by execution of O_4. Let us also assume that the VR-processor is currently executing VR_4 and it needs to obtain the value of Z assigned the latest before the object processor entered into O_4. The desired value is "200" and it is contained in the non-local variable section of VR-store-segment 2 since VR_3 has already been successfully completed. The non-local variable section of VR-store-segment 2 needs to be searched in order to get the desired value "200." The problem here is a little complex since if Z has not been assigned a value during execution of O_3, the desired value is "100" contained in the local variable section of VR-store-segment 1. It is not known in advance which VR-store-segment, between 1 and 2, contains the desired value. It is thus necessary to check VR-store-segment 2 first and if its non-local variable section does not contain Z, then the desired value is read from the location, in the local variable section of VR-store-segment 1, to which the logical address of Z is mapped.

In general, it is necessary to search ancestor VR-store-segments in the increasing order of their ages, until either an ancestor containing the variable in its non-local variable section is found or the ancestor addressed by a portion of the logical addresses of the variable is reached. Here employment of content-addressable memory modules for non-local variable sections of execution images is the key to the high speed of search. It enables simultaneous examination of non-local variable sections of all the "ancestor VR-store-segments" which are "descendants" of the one containing the variable in its local variable section. When the VR-processor issues a command for fetching the latest assigned value of a non-local variable among its values contained in ancestor VR-store-segments, one location-addressed memory module and possibly several content-addressable memory modules are simultaneously accessed. The location-addressed memory module is the one addressed by a portion of the logical address of the variable. The content-addressable memory modules are the ones containing the non-local variable sections of those ancestor VR-store-segments which are descendants of the one whose local variable section is mapped to the above location-addressed memory module. Selection of the latest assigned value among all the values of the variable retrieved from those memory modules is also a function of the duplex memory. With this duplex memory, the VR-processor can read or update any value contained in the VR-store with the amount of time close to one content-addressable memory cycle.

Resetting the system state when an execution image is evaluated to be unacceptable, is also speedy with this duplex memory. It is because the resetting process involves basically fetching the previous value of each non-local variable which has been assigned another value since the object processor entered into the object segment whose execution image turned out to be unacceptable, and then storing it into the location of the variable in the main working store. Each non-local variable which needs to be restored in the main working store to its previous value is identified by examining the non-local variable sections of the execution images produced since the object processor entered into the object segment whose execution image was rejected.

Local variable sections of VR-store-segments are mapped to location-addressed memory modules in the same manner as the main working store is mapped to location-addressed memory modules. For some VR-segments, each of their non-local variable sections may be mapped to an independent content-addressable memory module. For others, their non-local variable sections may co-exist inside the common content-addressable memory module, provided some form of an identification code is assigned to each section. The high speed requirement limits the size of each content-addressable memory module to a small one. It may sometimes be necessary to use more than one content-addressable memory module to hold the non-local variable section of a VR-store-segment.

It is believed that this architecture satisfactorily meets all the requirements mentioned before. Creating a copy of a state vector in this system does not incur any delay in processing object segments. The memory interference between the CP's executing object segments and VR-segments is absent or negligible. Employment of multiple content-addressable memory modules is believed to be an effective means of achieving the goals of low average access time and high store utilization. However, successful implementation of a system requires careful selection of design parameters concerning memory management.

AREAS OF EXTENSION AND FURTHER RESEARCH

The preceding sections dealt with the concept of failure-tolerant parallel programming and effective solutions to most fundamental problems involved in realizing its potential power. The model architecture devised for efficient execution of failure-tolerant parallel programs was specified at a highly abstract level in order to preserve simplicity and generality. It incorporated a minimal amount of facility. Naturally, the model architecture can be expanded in many directions to possess additional capabilities by incorporating various proven concepts and schemes. In addition, in order to put failure-tolerant parallel programming in general use, various program design and engineering tools need to be developed. Among numerous desirable extensions and research problems, only a few of the representative ones are listed below.

First, development of a language supporting failure-tolerant parallel programming is an immediate requirement. Such a language should contain more facilities for control structuring and data specification than the ones in conventional programming languages. No useful principles employed in other failure-tolerant program structuring schemes such as recovery block structuring,[14] need to be rejected in structuring failure-tolerant parallel programs. In view of the strong relationship between each object segment and the associated VR-segment, it is almost an indispensable requirement to put each pair of segments in a compartment in the program text. Such a compartment is called a *failure-tolerant segment.* More specifically, each VR-segment is directly dependent upon only one object segment in a failure-tolerant parallel program. Furthermore, the dependency structure among VR-segments is always the same as the dependency structure among object segments. The latter dictates the former. Thus the dependency among VR-segments as well as the dependency of a VR-segment on the associated object segment, need not be explicitly specified and should become a part of the definition of the failure-tolerant segment.

On the other hand, each object segment may be dependent on zero or more VR-segments which belong to other failure-tolerant segments. This type of dependency needs to be explicitly specified. This and other considerations mentioned in the previous section on desirable extensions of the state-of-art in failure-tolerant programming, led to the formulation of the model of the failure-tolerant segment depicted in Figure 7.

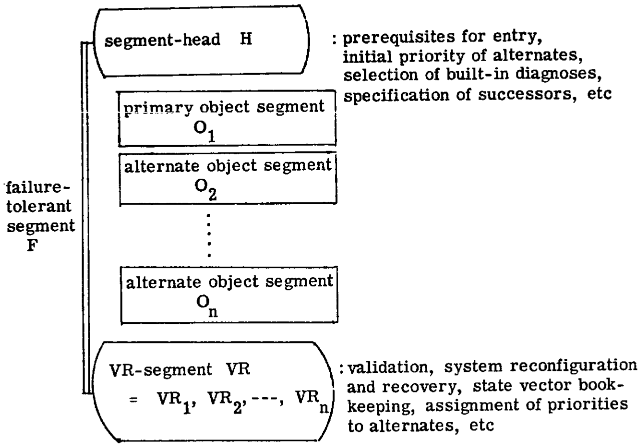

segment-head H
: prerequisites for entry, initial priority of alternates, selection of built-in diagnoses, specification of successors, etc
primary object segment O_1
alternate object segment O_2
failure-tolerant segment F
alternate object segment O_n
VR-segment VR $= VR_1, VR_2, ---, VR_n$
: validation, system reconfiguration and recovery, state vector book-keeping, assignment of priorities to alternates, etc

Figure 7—A structural model of a failure-tolerant segment

The model is a generalization of the model of the recovery block depicted in Figure 1. It consists of the *segment-head,* the primary object segment, several alternate object segments and the VR-segment. The segment-head specifies the prerequisite condition for entry into the (failure-tolerant) segment e.g., dependency on the completion of VR-segments belonging to other failure-tolerant segments. It specifies the *successor* failure-tolerant segments i.e. the ones whose initiation is dependent upon successful completion of the VR-segment. It also contains the declaration of the initial "execution priorities" of alternate object segments. It may also specify the types of abnormal conditions which may be recognized by the system during execution of object segments and the actions to be taken on occurrence of each condition e.g., "enter into the VR-segment," "record the occurrence and continue," etc. The VR-segment specifies the procedures of validation, VR-store management, and recovery including system reconfiguration and assignment of execution priorities to alternate object segments. The primary or alternate object segments can contain, nested within themselves, further failure-tolerant segments. However, the structuring rule illustrated in Figure 5 should become a part of the definition of the failure-tolerant segment.

Therefore, programming of the segment-head requires some special language primitives including JOIN-like primitive[9] used for specifying the prerequisite condition for entry, FORK-like primitive[9] for specifying the successors, ones for specifying types of abnormal conditions and appropriate treatments, etc. Programming of the VR-segment also requires some special language primitives such as ones for referring to the old values of variables, ones for system reconfiguration, etc. Development of a language containing all the facilities mentioned above is urgent.

Second, if parallelism among object segments is to be exploited, the model architecture and the program structuring scheme described so far needs to be generalized accordingly. Such a generalization is expected to be a gigantic task requiring a great deal of research.

Third, it is often necessary to periodically save verified state vectors into the file store either as the spontaneous action of the system or as controlled by the failure-tolerant program. Incorporation of an efficient filing capability into the model architecture is an essential requirement.

Fourth, in view of the high cost of a sizable content-addressable memory, it seems both necessary and desirable to use location-addressed modules as back-up memory when a certain program requires more space than that provided by available content-addressable modules. That is, additional content-addressable store space can be simulated on the basis of location-addressed modules and store structuring techniques such as hash-coding. Incorporation of the virtual memory into the model architecture will also be an interesting research subject.

CONCLUSION

The concept of failure-tolerant parallel programming was originated with the objective of utilizing extensive validation and recovery facilities at run-time without disturbing main-stream computation. The model architecture presented is believed to be a satisfactory solution to the efficient execution of failure-tolerant parallel programs. As further researches on the subjects mentioned in the preceding section progress, more insights will hopefully be gained into the potential power of failure-tolerant parallel programming and the cost-effective implementation of systems based on the model architecture.

ACKNOWLEDGMENT

This work was sponsored in part by the Joint Service Electronics Program under Air Force Contract F44620-71-C-0067 and in part by the National Science Foundation under Grant GJ-35833.

REFERENCES

1. Avizienis, A. et al., "The STAR (Self-testing and Repairing) Computers—An Investigation of the Theory and Practice of Fault-Tolerant Computer Design," *IEEE Trans. on Comp.*, November 1971, pp. 1312-1320.
2. Boyer, R. S. et al., "SELECT—A formal system for testing and debugging programs by symbolic execution," *Proc. 1975 Int'l Conf. on Reliable Software*, pp. 234-245.
3. Carter, W. C. and W. G. Bouricius, "A Survey of Fault-Tolerant Computer Architecture and its Evaluation," *Computer*, Jan.-Feb. 1971, pp. 9-16.
4. Chandy, K. M. and C. V. Ramamoorthy, "Rollback and Recovery Strategies for Computer Programs," *IEEE Trans. on Comp., February* 1972, pp. 137-146.
5. Chandy, K. M. et al., "Analytic Models for Rollback and Recovery Strategies in Data Base Systems," *IEEE Trans. on Software Engr.*, March 1975, pp. 100-110.
6. Chandy, K. M., "A Survey of Analytic Models of Rollback and Recovery Strategies," *Computer*, May 1975, pp. 40-47.
7. Chang, H. Y. et al., *Fault Diagnosis of Digital Systems*, Wiley-Interscience, 1970.
8. Connet, J. R. et al., "Software Defenses in Real-Time Control Systems," *Digest of the 1972 Int'l Symp. on Fault-Tolerant Computing*, pp. 94-99.
9. Conway, M., "A Multiprocessor System Design," *Proc. AFIPS Fall Joint Comp. Conf.*, pp. 139-146.
10. Dijkstra, E. W., "Structured Programming," in J. N. Buxton and B. Randell (eds.), *Software Engineering Techniques*, report on a Conf. sponsored by the NATO Science Committee, Rome, Italy, 1969, pp. 84-88.
11. Elemendorf, W. R., "Fault-Tolerant Programming," *Digest of the 1972 Int'l Symp. on Fault-Tolerant Computing*, pp. 79-83.
12. Elspas, B. et al., "An Assessment of Techniques for Proving Program Correctness," *Computing Surveys*, June 1972, pp. 97-147.
13. Hetzel, W. C. (ed), *Program Test Methods*, Prentice-Hall, 1973.
14. Horning, J. J. et al., *A Program Structure for Error Detection and Recovery*, Lecture notes in *Comp. Sci.* Vol. 16, Springer-Verlag, 1974, pp. 177-193.
15. Kennedy, P. J. and T. M. Quinn, "Recovery Strategies in the No. 2 Electronic Switching System," *Digest of the 1972 Int'l Symp. on Fault-Tolerant Computing*, pp. 165-169.
16. King, J. C., "A New Approach to Program Testing" *Proc. 1975 Int'l Conf. on Reliable Software*, pp. 228-233.
17. Kopetz, H., "Software Redundancy in Real-Time Systems," *Proc. IFIP Congress 1974*, pp. 182-186.
18. Pierce, W. H., *Failure-Tolerant Computer Design*, Academic Press, 1965.
19. Pikner, H., "Programmed Restarts," *Proc. Annual ACM Conf.*, 1971, pp. 13-27.
20. Ramamoorthy, C. V., R. C. Cheung and K. H. Kim, "Reliability and Integrity of Large Computer Programs," Lecture notes in *Comp. Sci.*, Vol. 12, Springer-Verlag, 1974, pp. 86-161.
21. Ramamoorthy, C. V. and K. H. Kim, "Software Monitors Aiding Systematic Testing and their Optimal Placement," *Proc. 1st Nat'l Conf. on Software Engr.*, pp. 21-26.
22. Randell, B., "System Structure for Software Fault Tolerance," *IEEE Trans. on Software Engr.*, June 1975, pp. 220-232.
23. Rohr, J. A., "STAREX-Self-Repair Routines: Software Recovery in the JPL-STAR Computer," *Digest of the 1973 Int'l Symp. on Fault-Tolerant Computing*, pp. 11-16.
24. Rohr, J. A., *System Software for a Fault-Tolerant Digital Computer* Ph.D. thesis, Dept. of Comp. Sci., Univ. of Ill. at Urbana-Champaign, 1973.
25. Short, R. A., "The Attainment of Reliable Digital Systems Through the Use of Redundancy—A Survey," *IEEE Comp. Group News*, March 1968, pp. 2-17.
26. Von Neumann, J., "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," in *Automata Studies, Annals of Math.* No. 34, Princeton Univ. Press, 1956, pp. 43-98.
27. Yau, S. S. and R. C. Cheng, "Design of Self-Checking Software," *Proceedings 1975 International Conference on Reliable Software*, pp. 450-457.

Strategic planning for MIS—A conceptual framework

by EPHRAIM R. McLEAN
University of California
Los Angeles, California

and

JOHN V. SODEN
McKinsey & Company, Inc.
Los Angeles, California

ABSTRACT

Increasingly, Management Information Services (MIS) divisions within organizations are becoming more centrally involved in the mainstream of corporate activities. MIS projects are becoming more complex and expensive; are affecting more aspects of the business; and are taking longer to design, develop, and install. For these reasons, effective planning for the MIS function is felt to be of paramount importance.

In recognition of this, a conference was held on Planning for MIS in 1974, jointly sponsored by McKinsey & Company, Inc., and the Graduate School of Management at the University of California, Los Angeles (UCLA). Attending were MIS executives and practitioners from both the public and private sectors. During the two days of discussions, a number of conclusions were reached which are detailed in this paper.

Building upon this conference, and on other work of the authors, the balance of the paper is devoted to presenting a framework for strategic planning for MIS. Definitions of key terms are included, as well as a discussion of some common planning pitfalls. Finally, a list of questions is given, designed to aid the MIS executive in carrying out his own organization's planning effort.

INTRODUCTION

Mark Twain's remark about the weather might well be applied to MIS planning: "Everyone talks about it, but nobody does anything about it." The importance of planning for improved managerial effectiveness is widely endorsed by practitioners and academics alike. In fact, given the accelerating pace of change in almost every aspect of the economy, planning is frequently touted as the key to success—if not to survival. But as with the weather, the gap between interest and achievement in the planning area is great. Faced with the pressing problems of day-to-day operations, many MIS executives have neither the time nor the inclination to invest in planning for the longer term. However, there are a number of major organizations here in the United States that *are* doing something about planning; in particular, planning for Management Information Systems (MIS). This paper is about such efforts.

The term, "MIS," standing for either management information *systems* or management information *services,* is being used increasingly throughout the world to refer to that cluster of activities which surround the computer and its supporting personnel. However, it is more than just the data processing department; for it includes the planning, analysis, and design activities—as well as the operational functions—which are necessary for effective computer-based information systems. For this reason, many MIS groups go under the broader title of "Management Services," incorporating not only the computer department but also operations research and management science staff specialists. In this paper, we will use "MIS" to reflect this broader set of activities—computer systems, management services, and indeed the organization itself which provides the foregoing.

In addressing the topic of planning for this group of activities, it is important to establish a proper perspective. As will be discussed later, planning can be looked at from both its time horizon and its focus. The former refers to whether it is short term (one to two years), medium term (two to five years), or long term (five years or more); and the latter, to whether it is focusing on strategic, managerial, or operational concerns. It is our intention here to concentrate on the strategic planning issues.

The reason for this choice is simple. As the MIS function assumes a more central role within organizations, it becomes vital that this role be properly planned for, so that it will be congruent with that of the overall organization. No longer is it feasible—if it ever were—to have systems for their own sake. This is a luxury no organization can afford. And if MIS is to be made responsive to larger corporate objectives, strategic long-range planning is essential.

THE McKINSEY-UCLA CONFERENCE

In recognition of the importance of this issue, an invitational conference was held in 1974 at the University of California, Los Angeles (UCLA), sponsored by McKinsey & Company, Inc., the management consulting firm, and the UCLA Graduate School of Management. This working conference was chaired by the authors of this paper with the assistance of Professor George A. Steiner of UCLA. Some two dozen MIS executives from major private and public sector organizations participated in the two-day conference. A survey was administered to them, focusing on the objectives, development process, and end products of their individual long-range planning efforts.[1] Also, these executives participated in extensive discussions regarding various aspects of their own planning experience as well as preparing summaries of the long-range planning activities of their own organizations.*

The theme of the conference was focused on those aspects of planning that had to do with the central issues of the information services organization itself, as opposed to the planning for individual information systems projects. The reasons for this focus were twofold. First, we wanted the conference to have a broad managerial orientation rather than a technical one. In this way, we believed that the results would be of interest not only to MIS managers and practitioners but also to general corporate executives as well.

The second reason for our choice was the relatively virgin nature of the MIS planning field. Had we chosen to look at the planning problems associated with the design and installation of specific information systems, we would have been addressing an area in which much work has already been done, with articles and books on project and systems management in abundance. However, the literature on strategic and long-range planning for information services is fairly sparse.**

In order to obtain a good cross section of various approaches to planning, the conference participants were chosen to represent a wide variety of private companies, as well as government and education enterprises at the local, state, and federal levels. The average participant represented an organization that had annual revenues or total budget expenses greater than $1 billion, had an annual MIS budget of over $15 million, and had been carrying out a formal MIS long-range planning effort for more than three years. These participants, therefore, represented relatively large, mature MIS organizations, experienced in planning for the information systems effort.

The following are the major points which emerged from the conference discussions.

1. There is a growing need for more formal long-range information systems planning as systems become more complex, require longer to develop, involve multiple functions or departments, cost more money, and have greater competitive impact.
2. The benefits from long-range planning—improved short-term decision making, enhanced communication with both top management and user groups, and a firmer grasp on resource commitments—generally outweigh the costs of the undertaking.
3. Formal planning approaches range from the "controlled reaction" tactics of formally evaluating and ranking known project ideas, to the strategic "top-down" scanning for high-potential application opportunities within the context of the overall organization's strategic plan.
4. The selection of a particular planning approach requires a careful balancing of factors such as the role and charter of the MIS organization, its degree of maturity, and the sophistication of the overall company and individual "user" executives.
5. Success in planning for information systems hinges on three factors:
    a. The previous credibility of the MIS group in managing new project development and ongoing computer operations.
    b. The maturity of the overall organization's management processes, particularly with regards to conducting business planning and in making capital allocation decisions.
    c. The choice of a particular MIS planning approach which suits the needs and constraints of the particular organization at that point in time.
6. In those organizations most advanced in their planning, the MIS executives have become an integral part of the management team of their organization; and, in these companies, MIS strategies have a major impact on, and a corresponding interrelationship with, the long-range business plans of the enterprise. In other words, MIS planning is "interactive," not "reactive."
7. If an MIS organization is relatively underdeveloped in terms of standards, computer operations effectiveness, individual project management capability, and the like, it would be well advised to concentrate on the short term, severely limiting concern for the long term until the near-term situation has been substantially improved.
8. Good formal planning must complement, but can-

* These papers, together with other chapters on MIS planning, are to be published shortly.[2]

** For a few noteworthy exceptions, see References 3 through 12.

not replace, the political sensitivity, entrepreneurship, conceptual contribution, and basic business leadership required of the successful MIS executive.

STRATEGIC AND LONG-RANGE PLANNING

In addition to the preceding findings, another item came to light which is worthy of mention. Because of the multi-year planning horizon of most computer-related projects, it is easy to equate long-range planning with any planning effort which has a horizon greater than one year. And thus, fundamental questions such as "Where is the information services organization going?" and "How is it contributing to the overall success of the enterprise?" become confused with "What project should be started next?" and "How can the continued development of existing projects be more effectively coordinated?"

Unfortunately, the term "Long-Range Planning"—which was the title of the conference—does little to sharpen this distinction. For some conference participants it meant focusing on the former questions; for others, on the latter ones. This dual interpretation became evident from both the conference discussions and the papers which the participants prepared. It can be argued, of course, that both uses of the term are equally valid and equally important for the success of the information services organization. Certainly, if the ongoing operations and project development is not effectively planned and managed, it does little good to speculate on where the organization will be five to seven years in the future. More than likely, it will be an organization with a new cast of characters!

However, as discussed earlier, we wanted the conference to focus on the central issues of setting organizational objectives and deciding upon appropriate strategies and policies. Thus, it seems in retrospect that "Strategic Planning" would have been a more appropriate title than "Long-Range Planning."

A STRATEGIC PLANNING FRAMEWORK

Based upon the discussions of the conference, as well as on other research and on the industrial and governmental experience of the authors, an MIS strategic planning framework has been devised (see Figure 1). The left half of the diagram portrays the tasks needed to arrive at the MIS objectives, strategies, and policies. The right half notes the tasks necessary to accomplish

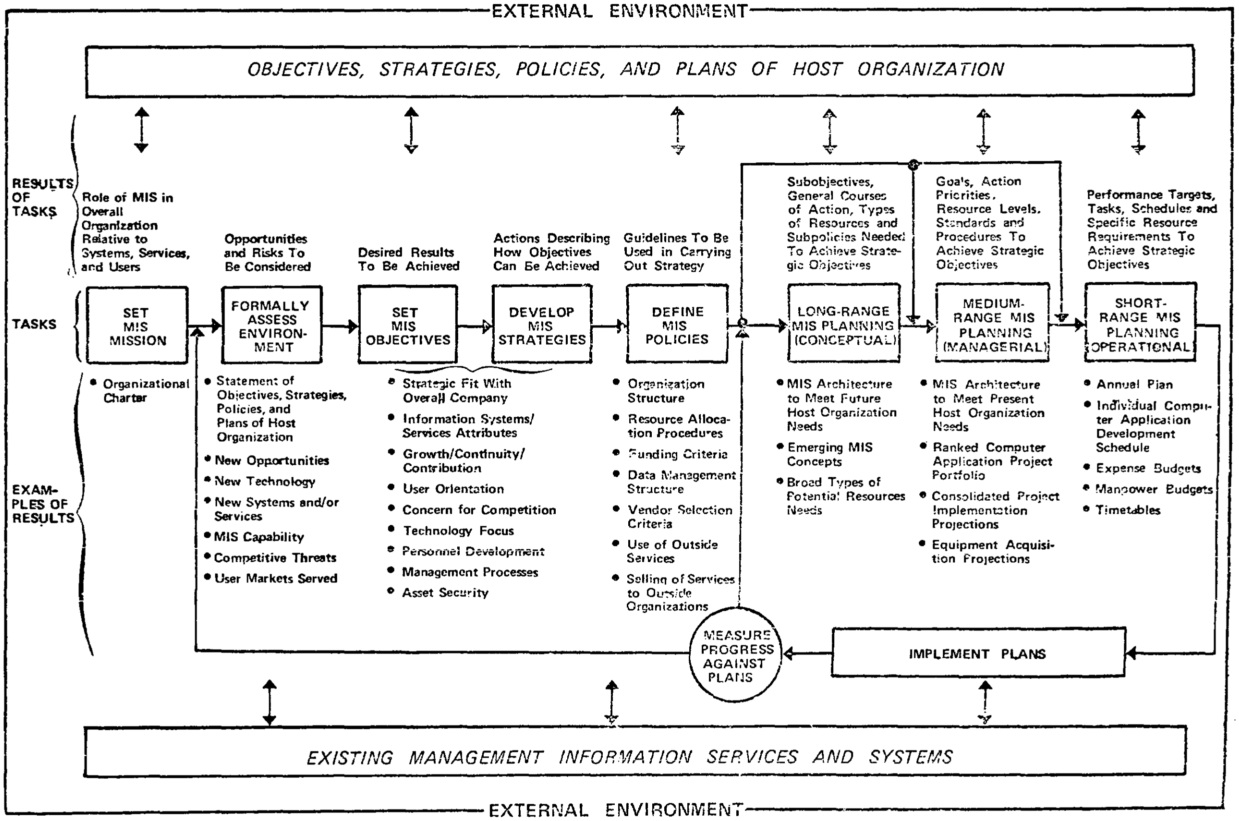

EXTERNAL ENVIRONMENT
OBJECTIVES, STRATEGIES, POLICIES, AND PLANS OF HOST ORGANIZATION
RESULTS OF TASKS
Role of MIS in Overall Organization Relative to Systems, Services, and Users
Opportunities and Risks To Be Considered
Desired Results To Be Achieved
Actions Describing How Objectives Can Be Achieved
Guidelines To Be Used in Carrying Out Strategy
Subobjectives, General Courses of Action, Types of Resources and Subpolicies Needed To Achieve Strategic Objectives
Goals, Action Priorities, Resource Levels, Standards and Procedures To Achieve Strategic Objectives
Performance Targets, Tasks, Schedules and Specific Resource Requirements To Achieve Strategic Objectives
TASKS
SET MIS MISSION
FORMALLY ASSESS ENVIRON-MENT
SET MIS OBJECTIVES
DEVELOP MIS STRATEGIES
DEFINE MIS POLICIES
LONG-RANGE MIS PLANNING (CONCEPTUAL)
MEDIUM-RANGE MIS PLANNING (MANAGERIAL)
SHORT-RANGE MIS PLANNING (OPERATIONAL)
EXAM-PLES OF RESULTS
• Organizational Charter
• Statement of Objectives, Strategies, Policies, and Plans of Host Organization
• New Opportunities
• New Technology
• New Systems and/or Services
• MIS Capability
• Competitive Threats
• User Markets Served
• Strategic Fit With Overall Company
• Information Systems/Services Attributes
• Growth/Continuity/Contribution
• User Orientation
• Concern for Competition
• Technology Focus
• Personnel Development
• Management Processes
• Asset Security
• Organization Structure
• Resource Allocation Procedures
• Funding Criteria
• Data Management Structure
• Vendor Selection Criteria
• Use of Outside Services
• Selling of Services to Outside Organizations
• MIS Architecture to Meet Future Host Organization Needs
• Emerging MIS Concepts
• Broad Types of Potential Resources Needs
• MIS Architecture to Meet Present Host Organization Needs
• Ranked Computer Application Project Portfolio
• Consolidated Project Implementation Projections
• Equipment Acquisition Projections
• Annual Plan
• Individual Computer Application Development Schedule
• Expense Budgets
• Manpower Budgets
• Timetables
MEASURE PROGRESS AGAINST PLANS
IMPLEMENT PLANS
EXISTING MANAGEMENT INFORMATION SERVICES AND SYSTEMS
EXTERNAL ENVIRONMENT

Figure 1—MIS strategic planning framework

the more detailed planning efforts within the long-range, medium-range, and short-range time frames. First, however, it is necessary that we define what we mean by "strategic planning."

Strategic MIS planning, like Robert Anthony's definition of corporate strategic planning,[13] is the process of deciding on objectives for the MIS organization; on changes in these objectives; on the resources used to obtain these objectives; and on the policies that are to govern the acquisition, use, and disposition of these resources. Strategic MIS planning typically occurs at infrequent intervals and is often triggered by the need for an enterprise to resolve a particularly substantive issue or issues that involves the MIS entity.

Strategic planning tasks

Set the MIS mission

The first task in strategic planning, as shown in Figure 1, is to set the MIS mission; that is, to define the charter of the information services organization. This broad definition of organizational role must naturally be done within the mission and purpose of the overall organization of which MIS is a part. Sometimes the MIS organization receives this mission as given; other times it is arrived at through mutual discussion with top management.

Assess the MIS environment

Once this mission is set, the next task is to assess the MIS environment—to consider the opportunities and risks which are present now and might be present in the future. This would include consideration of such things as:

1. The objectives, strategies, policies, and plans of the host organization.
2. The competitive position of the overall organization.
3. The user groups within the organization—their needs, their current use of MIS resources, and their perceptions of the capability of the MIS organization.
4. The present and emerging technology for information processing.
5. The ability of the MIS organization to effect change.

Set the MIS objectives and develop the strategies

With the MIS mission established and a thorough appraisal made of the environment, it next becomes necessary to set the MIS objectives. There are the desired results that are to be achieved by the MIS organization. Closely linked with the statement of objectives is the development of the strategies or broad courses of action that will be needed in order to achieve these objectives. Thus, objectives and strategies are intimately interwoven; consideration of one invariably involves consideration of the other. These objectives, and their accompanying strategies, typically deal with the following types of items:

1. The fit of the MIS objectives within the overall organizational objectives.
2. The growth, continuity, and level of contribution of the MIS function within the organization.
3. The classes and types of systems and services to be offered.
4. The role of users in systems development efforts.
5. The types of technology to be employed.
6. The type of management and staff to be developed.
7. The posture of the MIS organization vis-à-vis the user, the host organization, the competitive environment, and the professional milieu.

Define the MIS policies

The determination of policies is a critical aspect of strategic planning. Policies are the guidelines to be used in carrying out the strategy. They are specific statements that cover such things as the internal organizational structure of the MIS division; the criteria to be used in deciding upon overall funding levels and resource allocations; the use of steering committees; the procedures to be used in selecting vendors, buying outside services, and/or selling services to outside users; the employment of a data base management scheme; and so forth.

Setting the policy for how the company is going to decide how much to spend for MIS is seldom directly addressed, and it is an area of particular frustration for many companies. Sometimes a mixture of techniques is decided upon as a means of arriving at the MIS budget. These include, for example, a fixed percent of sales or assets; comparable expenditures of similar companies, adjusted for size and profitability; whatever the major company profit centers will agree to pay for; base amounts, plus discretionary increases for high-potential projects; and/or the amount spent last year plus adjustments for inflation. All too often, a lack of policy in this area leads the MIS group to concentrate its attention on justifying the acquisition of major new computer equipment without giving much concern as to underlying reasons for such equipment.

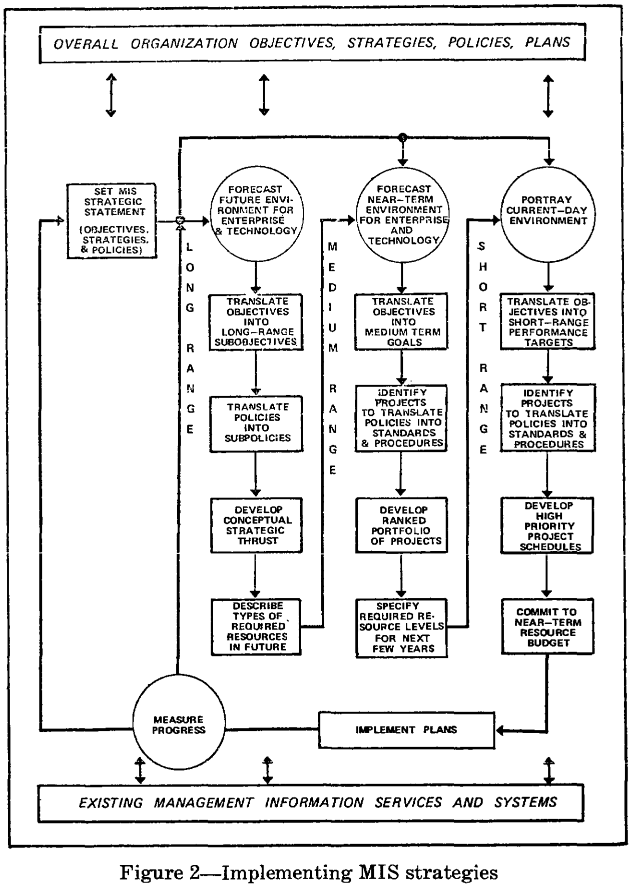
Policies with regard to the allocation of scarce MIS resources are particularly critical since they provide the guidelines by which the portfolio of current and future projects will be selected, funded, and managed. Unfortunately, in many companies where these resource allocation policies have not been well established, MIS management has little incentive to perform disciplined economic evaluations of new project proposals, and general corporate policies are often not precise enough to be useful in evaluating these proposed projects. Thus, in many instances, the implicit resource allocation policy frequently becomes one of allowing the MIS division the self-indulgence of selecting projects primarily based on its interest in utilizing the latest in information processing technology.

Although such policy setting activity is admittedly difficult, it is essential that it be done in order that subsequent, more detailed planning efforts may have a better chance of success.

Planning to implement MIS strategies

As shown in Figure 2, the plans which are needed to implement MIS strategies can be of several types, each of which has the goal of translating the MIS objectives and strategies into increasingly more detailed and specific plans.

Long-range MIS planning deals with meeting the *future* MIS needs of the host organization. It is largely conceptual in character and can have a horizon of from five to seven years or longer. It does not deal with specific projects or even groups of projects, but with emerging types of user needs and approaches that might be useful in addressing these needs. It must also plan for the organizational philosophy to guide the MIS organization of the future and for the skills and capabilities that will be needed in developing and managing future systems. An example of this type of plan is the information systems design architectures that are being developed within a number of organizations.



OVERALL ORGANIZATION OBJECTIVES, STRATEGIES, POLICIES, PLANS
SET MIS STRATEGIC STATEMENT (OBJECTIVES, STRATEGIES, & POLICIES)
LONG RANGE
FORECAST FUTURE ENVIRONMENT FOR ENTERPRISE & TECHNOLOGY
TRANSLATE OBJECTIVES INTO LONG-RANGE SUBOBJECTIVES
TRANSLATE POLICIES INTO SUBPOLICIES
DEVELOP CONCEPTUAL STRATEGIC THRUST
DESCRIBE TYPES OF REQUIRED RESOURCES IN FUTURE
MEDIUM RANGE
FORECAST NEAR-TERM ENVIRONMENT FOR ENTERPRISE AND TECHNOLOGY
TRANSLATE OBJECTIVES INTO MEDIUM TERM GOALS
IDENTIFY PROJECTS TO TRANSLATE POLICIES INTO STANDARDS & PROCEDURES
DEVELOP RANKED PORTFOLIO OF PROJECTS
SPECIFY REQUIRED RESOURCE LEVELS FOR NEXT FEW YEARS
SHORT RANGE
PORTRAY CURRENT-DAY ENVIRONMENT
TRANSLATE OBJECTIVES INTO SHORT-RANGE PERFORMANCE TARGETS
IDENTIFY PROJECTS TO TRANSLATE POLICIES INTO STANDARDS & PROCEDURES
DEVELOP HIGH PRIORITY PROJECT SCHEDULES
COMMIT TO NEAR-TERM RESOURCE BUDGET
MEASURE PROGRESS
IMPLEMENT PLANS
EXISTING MANAGEMENT INFORMATION SERVICES AND SYSTEMS

Figure 2—Implementing MIS strategies

Medium-range MIS planning is what many organizations call their long-range plan. It is the planning that is necessary to meet the host organization's *present* MIS needs, projected two or five years into the future. It is a portfolio of projects, ranked by importance, coupled with projections for their implementation. It also involves the planning for hardware and software acquisitions and conversions, and for the staffing of multi-year projects and development activities.

Short-range MIS planning is generally equivalent to the MIS annual plan. It involves detailed budget preparation, manpower scheduling, and the creation of timetables for individual projects. It also often includes quantitative statements regarding performance targets for the MIS group. It is relatively operational in character.

The choice of a particular approach or set of approaches to MIS planning, assuming that the strategic planning has been properly done, is particularly important. Many organizations falter in that they attempt to carry out all three types of MIS planning simultaneously, before mastering the intricacies of the short-term plans necessary for the effective management of present activities.

Ideally, the MIS strategy should be translated into current-day decisions by sequentially developing first the long-range conceptual plan, then the medium-range managerial plan, and finally, the short-range operational plan. However, those companies that have very short-term MIS strategic objectives, such as to build credibility with users or to "get away from the crocodiles," should focus first on simple operational plans that include very specific quantitative performance measures for effectiveness and efficiency. Then, once the MIS organization has mastered its short-term challenges, it can extend its planning horizon.

We believe that a majority of companies would find a major investment in long-range conceptual planning for MIS to be of little value, for it appears that most enterprises have not yet mastered medium- and short-range planning. However, since short-range operational planning is relatively straightforward, we will focus our remaining discussion on medium-range planning. The development of medium-range MIS plans generally involves the following activities.

1. *Identify potential projects.* Fill a "hopper of opportunity" with ideas for projects relating to new computer system development efforts, enhancements to existing computer services, up-

grading of ongoing MIS activities, and strengthening of the MIS organization and personnel.

2. *Evaluate and rank projects by priority.* Filter the contents of this hopper of opportunity by using the resource allocation policies and overall MIS objectives previously established, so as both to select those projects that will be undertaken next and to identify the types of projects that might be undertaken in future years.
3. *Translate projects into time-phased profiles of activities, resource requirements, and action steps.* Define the means, requirements, and plans of action for implementing the selected MIS projects.

Identifying potential projects

The first step in the development of an information systems plan is to identify those information system projects that have high potential for the organization. There is a wide variety of approaches that can be used to scan for these new opportunities for system development. The following are some of the most common.

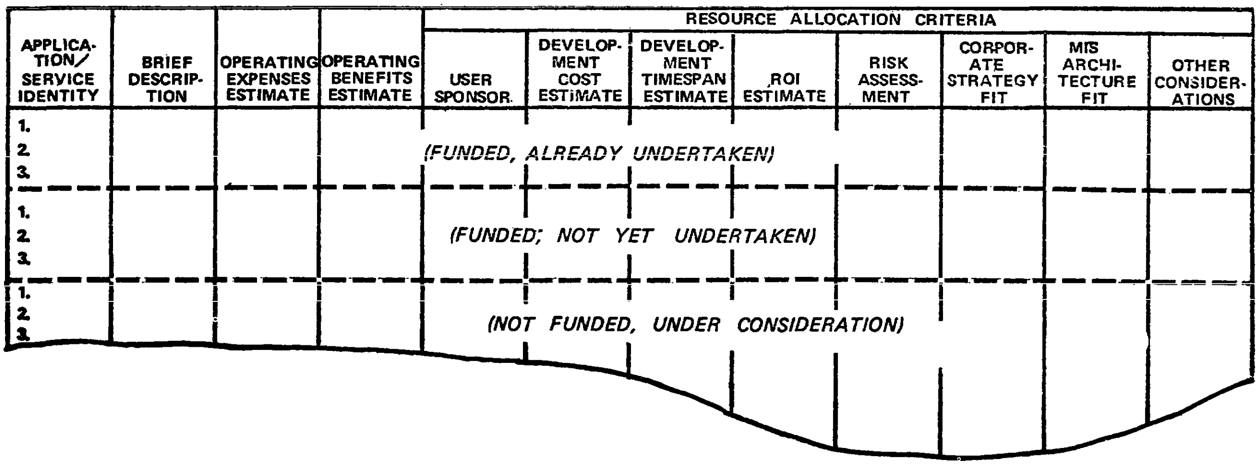
1. *The isolated approach,* where the MIS organization reacts to environmental factors by modifying existing systems to meet new legal requirements; proposes retrofits to existing computer systems to enhance their cost and/or operating characteristics; operates on the basis of its own "intuition" regarding new opportunities; and/or synthesizes a list of requests for changes to existing systems already submitted by user managers.
2. *The emulative approach,* where the MIS organization picks up ideas for new projects from the successful computer systems of other companies, similar in size or industry group.
3. *The bottom-up approach,* in which MIS systems analysts interview either selected user executives or, in certain company situations, all major user groups so as to identify the major decision areas, possible information gaps, and operating inefficiencies which could be improved through the use of better computer systems.
4. *The reactive approach,* in which the MIS organization simply responds to decisions made by either the chief executive officer or some higher level corporate executive as to which are the most appropriate new computer system projects for the MIS division to undertake.
5. *The derived top-down approach,* used when the host organization has no overall strategy, involves a detailed analysis of the company in order to hypothesize an overall corporate business strategy from which new high-priority computer system ideas can be selectively developed. These ideas are then further refined through interviews and appropriate systems analysis.
6. *The top-down approach,* in which the MIS organization develops new services as an outgrowth of the existing, substantive company business plan.
7. *The interactive approach,* in which the MIS organization interacts with other parts of the business during the normal company planning cycle so that the identification of MIS project ideas, and assessment of their likely business impact, are integrated into the planning activities of management throughout the company.

The important task here for the MIS executive is to select the particular approach or combination of approaches that best fits the company's unique needs at that point in time.

Project selection

Invariably, there comes a point in this planning process when the collected ideas for new projects must be sorted out so that the highest priority applications or services can be undertaken in the near term. A problem of many planning efforts is the failure to perceive, in advance, the need to establish a means for conducting such a screening. Since the MIS effort is—or should be—a service function to the entire enterprise, this sorting out of priorities is difficult, even when well-chosen resource allocation policies have been established. A simple chart to aid in carrying out this ranking is shown in Figure 3, where the project ideas are listed, and the previously selected resource allocation criteria are applied to each project to the maximum extent possible. The major challenge here is to obtain a summary evaluation of these projects without entering into a detailed project feasibility study, since the primary objective of this effort is to decide just which projects should initially be allocated feasibility study funds.

To make these difficult priority rankings, many corporations have established computer applications steering committees in which judgments regarding these alternate investments in new systems are made on an ongoing basis by selected members of top management. In this way, a continuing consensus regarding the need for new system development activities is obtained; and, in cases where trade-offs and compromises must be made, the affected individuals and user organizations have an active voice in the decisions. Unfortunately, many of these committees have not been as successful in carrying out their prescribed role as they might have been. The overriding problem with ineffectual applications steering committees is that they are unable to resolve the many competing demands for limited MIS resources. The committee either becomes a "rubber stamp" for the recommendations of the MIS director or a "buck passer," deferring any hard choices to a higher level of management. More often than not,



				RESOURCE ALLOCATION CRITERIA							
APPLICATION/ SERVICE IDENTITY	BRIEF DESCRIPTION	OPERATING EXPENSES ESTIMATE	OPERATING BENEFITS ESTIMATE	USER SPONSOR	DEVELOPMENT COST ESTIMATE	DEVELOPMENT TIMESPAN ESTIMATE	ROI ESTIMATE	RISK ASSESSMENT	CORPORATE STRATEGY FIT	MIS ARCHITECTURE FIT	OTHER CONSIDERATIONS
1. 2. 3.				(FUNDED, ALREADY UNDERTAKEN)							
1. 2. 3.				(FUNDED; NOT YET UNDERTAKEN)							
1. 2. 3.				(NOT FUNDED, UNDER CONSIDERATION)							

Figure 3—Sample project portfolio overview

this failure is due to the absence of a clearcut resource allocation policy, one which provides unambiguous guidelines for choosing among projects—guidelines which are firmly rooted in business considerations, not technical ones.

Develop action plans

If an appropriate approach to scanning and ranking information system development projects is taken, the third step in planning—the development of associated resources and activity schedules—is somewhat more mechanical in nature, although vital in reaching such important decisions as computer equipment selection. Generally, these project schedules are quite precise for the near term and somewhat more general for future years. This is as it should be, since the primary purpose of the planning effort is to make near-term decisions that are consistent with a longer term direction, not to decide on specific computer system projects that are to be undertaken three or four years in the future.

FEEDBACK

It is important to recognize that the planning process and resulting decisions are dynamic, not static. As the bottom part of Figures 1 and 2 indicate, there is an important feedback loop, one which measures progress against plans, and ultimately, against the objectives and strategies themselves. For nothing should be felt to be fixed or "cast in concrete." Many an MIS executive has wailed "But this isn't the way we planned things last year" forgetting that a plan is not a forecast *of* the future, but a way of being better prepared *for* the future. Plans and strategies should be flexible and able to be modified and changed as circumstances dictate.

CONCLUSION

Although conceptual frameworks, such as the one that we have presented here in this paper, can be helpful to the MIS executive, it can never replace the common sense and good judgment that an experienced manager must possess. The first step that such a manager must take in reassessing the MIS planning efforts of his own organization is to recognize that the MIS function is not an end in itself but a part—and hopefully a vital part—of the larger objectives and activities of the overall enterprise. Then, in considering how to launch a new and/or revised planning effort, the MIS executive should seek to answer the following questions:

1. Are we reasonably adept at estimating costs, benefits, and risks of proposed new computer projects?
2. In general, do our MIS project postimplementation audits indicate that the MIS organization was able to develop projects within cost and timetable estimates, and that users were able to achieve the benefits they committed to at the outset of the project?
3. Is the MIS function now operating above average from an effectiveness and efficiency standpoint?
4. What approaches are we taking to scan for new computer investment opportunities within the organization?
5. Do we have agreed-upon documented objectives, broad strategies, and policies for the MIS function?
6. Are we making sure that the MIS plan will focus on the company's—rather than the MIS division's—use of the computer?
7. Are the MIS strategy and plans integrated with our overall corporate strategy and plans?

8. What are the respective roles of top management, the MIS division, and the users groups within our organization in the launching and conduct of the planning effort?
9. What will be the end product of this planning effort?
10. Are we monitoring the planning process itself so as to be in a position to improve our efforts the next time around?

The answers to these questions should provide not only a profile of the current status of MIS planning within an organization; but where the responses are negative—or vague—they should point the way to where beneficial progress can best be made.

REFERENCES

1. Soden, John V. and Charles C. Tucker, "Long-Range MIS Planning Practices," *Journal of Systems Management*, (forthcoming).
2. McLean, Ephraim R. and John V. Soden, (Editors), *Strategic Planning for MIS*, Wiley Interscience, New York, 1976.
3. Albrecht, Leon K., *Organization and Management of Information and Management of Information Processing Systems*, The Macmillian Company, New York, 1973.
4. Blumenthal, Sherman C., *Management Information Systems: A Framework for Planning and Development*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1969.
5. Fried, Louis, "Long-Range Planning for EDP," *Auerbach Data Processing Management Reports*, 1974, 16 pp.
6. Kriebel, Charles H., "The Strategic Dimension of Computer Systems Planning," *Long Range Planning*, September 1968, pp. 7-12.
7. McFarlan, F. Warren, "Problems in Planning the Information System," *Harvard Business Review*, March-April 1971, pp. 75-89.
8. McFarlan, F. Warren, Richard L. Nolan and David P. Norton, *Information Systems Administration*, Holt, Rinehart and Winston, Inc., New York, 1973.
9. Schwartz, M. H., "MIS Planning," *Datamation*, September 1, 1970, pp. 28-31.
10. Siegal, Paul, *Strategic Planning of Management Information Systems*, Petrocelli Books, New York, 1975.
11. Soden, John V. and George M. Crandell, "Practical Guidelines for EDP Long-Range Planning," *National Computer Conference Proceedings*, 1975, pp. 675-679.
12. Young, Richard C., "Systems and Data Processing Departments Need Long-Range Planning," *Computers and Automation*, May 1967, pp. 30-33, 45.
13. Anthony, Robert N., *Planning and Control Systems: A Framework for Analysis*, Boston Graduate School of Business Administration, Harvard University, 1965.

The economics of software quality assurance

by DAVID S. ALBERTS
The Mitre Corporation
McLean, Virginia

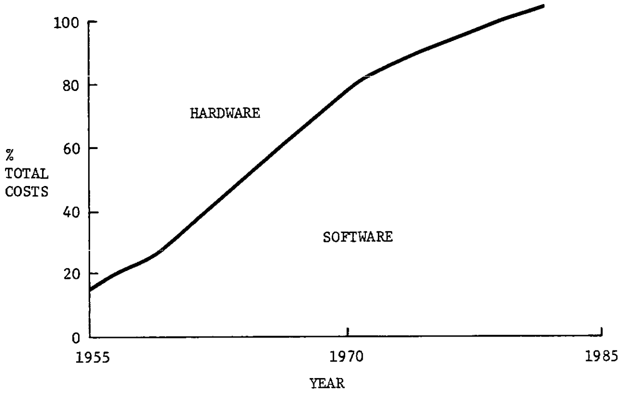
ABSTRACT

This paper presents an examination into the economics of software quality assurance. An analysis of the software life-cycle is performed to determine where in the cycle the application of quality assurance techniques would be most beneficial. The number and types of errors occurring at various phases of the software life-cycle are estimated. A variety of approaches in increasing software quality (including Structured Programming, Top Down Design, Programmer Management Techniques and Automated Tools) are reviewed and their potential impact on quality and costs are examined.

INTRODUCTION

Current realities of large scale computer systems have provided the impetus to undertake this examination into the need for and potential of a quality assurance program. Proponents of quality assurance claim that significant savings in both cost and time can be achieved in addition to improved system performance if a quality assurance program is implemented. The purpose of this paper is to examine these claims by addressing the economics of software quality assurance. Software rather than hardware is the subject of the analysis since the costs of software have far outstripped the costs of hardware and the trend seems to be continuing in this direction (Figure 1).

The stakes involved are high. Estimates of recent Air Force annual expenditures on software are over $1 Billion.[56] WWMCCS alone was estimated to involve $¾ Billion for software (about 10 times its hardware costs),[2] while major software systems also run into hundreds of millions of dollars (IBM OS/360 $200M,[11] SAGE $250M[11] and NASA manned space program $1B[12]). Indirect costs must be added to these huge sums and are by no means trivial in of themselves. For example, software delays often cause delays in reaching the operational phase of a system's life. A 6-month delay (considered almost on-time) translates into a $100M loss of services, based upon a projected 7 year operational life and a $1.4 billion project.



SOURCE: 2

Figure 1—Importance of controlling the cost/effectiveness of S/W

The actions which could be undertaken under the umbrella of a quality assurance plan are quite diverse; so diverse that it is difficult to separate these actions from project management. However, quality assurance is only one aspect of project management. First this analysis addresses the software life cycle and the relative cost on each portion of the cycle. Next productivity is considered insofar as the reduction in errors in each portion of the cycle impacts cost. Finally a variety of methods, techniques and tools which can directly affect the error rate/severity experienced will be examined.

Two questions drive this analysis. First, can QA work? and second, Is it worth it? This paper brings together the experiences and thoughts currently in circulation and forms these into an analysis of the issues involved and presents composite estimates of the potential target of quality assurance (cost of error) and the reported experience of quality assurance programs and methods currently available. Because of the difficulty in separating QA methods from project management and the absence of good cost accounting standards, the costs of a quality assurance program are not explicitly treated in this paper.

PHASES OF THE SOFTWARE LIFE CYCLE

To ensure a complete and systematic review of the potential for a QA plan, each aspect of the software "life cycle" will be examined to determine at what point QA can support substantial improvements. Both direct and indirect costs will be considered.

Direct costs are those associated with the actual performance of the particular phase of the software life cycle under consideration while indirect costs include schedule slippages, system degradation, and errors which contribute or add to the cost of subsequent stages in the process. The software life cycle can be broken down into four phases: Conceptual, Requirement, Development and Operations. While other authors have broken this cycle down somewhat differently, either by separating *Development* into two or more separate phases or by extending *Requirements* to include part of the development phase, the categorization shown here more closely corresponds to distinct levels of effort or expenditures.

After a brief qualitative discussion of the potential role of QA in each of the four phases of the software life cycle, the amount of time and relative costs incurred in the performance of each of these phases will be reviewed. Available data on contributions of errors to cost and delay is examined later.

The conceptual phase

This phase begins with the recognition of a need for the system. The feasibility and general worthiness of a proposed system is addressed. Usually a management decision is required to move into the next phase which involves more detailed specifications of performance characteristics. This phase is typified by numerous briefings designed to establish a recognized need for and cost/effectiveness of the system vis-a-vis organizational missions and functions. Order of magnitude cost figures are the typical modus operandi.

This phase has a relatively low contribution to total cost and may last several years. The question of *software* quality assurance is essentially moot throughout this phase of the life cycle. However, the role of software as it may interface with hardware, and gross estimates of costs and schedules should be reviewed as part of a larger quality assurance effort.

Failure to adequately address these issues could result in having to incorporate into the software development functions or design features which could have been accomplished better in other ways and which restrict flexibility or increase the complexity of the software.

The requirements phase

This phase of the software life cycle refines the conceptual system, further delineating the functions and interplay between hardware, software and the user. In general, data inputs and system outputs are specified and overall load and performance characteristics are determined. In many cases, specific determinations of system hardware and user-oriented languages are made. A properly designed Request for Proposal (RFP), even if the system is to be done inhouse (this step in the design process is skipped only at considerable risk), treads a thin line between overspecification and insufficient detail. The former is often caused by past contractor failures while the latter is a reflection of the fact that the user simply does not know what he really wants or needs.

To a large extent, the "die is cast" with the issuance of an RFP (or corresponding internal document). The constraints placed on system performance, hardware and software at this early stage of the life cycle can have enormous repercussions on the flexibility, reliability, maintainability and cost of the system. Implicit trade-offs between system throughput and ease and cost of use, enhancement and maintenance are often made.

Realistically one cannot expect a prospective vendor to do the necessary work required to examine and weigh each of the possible solutions to the design problem. Even with the most competent of vendors, their objective function differs from the clients. Specifically, a vendor's staff may have certain backgrounds and expertise, or his equipment characteristics more adaptable to one family of solutions than another. To save time or money a vendor may modify an already developed product or assemble a patch work of available system modules rather than seek an "optimal" solution.

Thus, quality assurance cannot begin any later than this phase without considerable risk. The phases which follow are characterized by much higher expenditures than these first two phases, with the obvious result that errors carried forward from this point are very costly.

The development phase

This phase is a transitional one bridging the gap between a well defined concept and an implementable system. The big black box between inputs and outputs has to be broken down into programmable units, logic determined and finally coded. The testing and validation tasks require the generation of test data and test parameters and the development of test tools. Documentation provides the vital link to connect the test activities to the designers, programmers and coders.

It is during this stage that a QA activity reaches its peak, for with increasing detail and concreteness comes the need for constant monitoring to assure that the system in reality is the system in concept. Quality assurance in this phase is simultaneously concerned with the correctness of (1) functional requirements, (2) detailed design, (3) program logic, and (4) code. In addition, the specificity and clarity of the documentation is also a proper subject of a QA plan. The testing and validation of the system or the "quality control" function is the most viable aspect of a quality assurance plan. For many developers, all too often, it *is* the QA plan. This tendency is to become lost in code is at the risk of deviations from intended system functions. Correctness of code is not a guarantee that the code is doing what the user required, but rather that is doing what it was designed to do; quite a different matter. Errors in design are far more perfidious than errors in logic.

The operations phase

This phase bridges the gap between the developers and the users. If QA proponents are correct, some payoff attributed to QA should be noticed during the implementation part of this phase, but its greatest contribution will appear during the productive part of the life cycle which is oddly called "maintenance." This terminology may be an indication of the general lack of quality assurance which exists.

More often than expected, the implementation period becomes a "field test" with the essential aspects of the development phase extending far into the operations phase. Design or even worse functional errors are frequently uncovered which may require extensive reprogramming. The start of implementation is often merely an artificial contrivance to cover a scheduled deadline rather than at the completion of the development phase.

THE SHAPE OF THE SOFTWARE LIFE CYCLE

To place the various aspects of the quality assurance function into perspective, it is necessary to look at the relative costs and time requirements of each of the phases described in the previous section. Figure 2 represents the idealized shape of the software life cycle.[28,13,38,39,40,45] While actual project experience is difficult to come by, a search of the literature for real-world cost and time data has been sufficiently productive to enable the construction of a composite software life cycle. This composite was developed from bits and pieces of available information on different phases and parts of phases of large systems. The degree of consistency found among projects gives rise to a fair degree of confidence that this composite is a useful tool in obtaining estimates of the potential benefits of a QA plan. Using the composite life cycle concept, this section relates the time required for accomplishing each portion of the cycle to software costs expended.

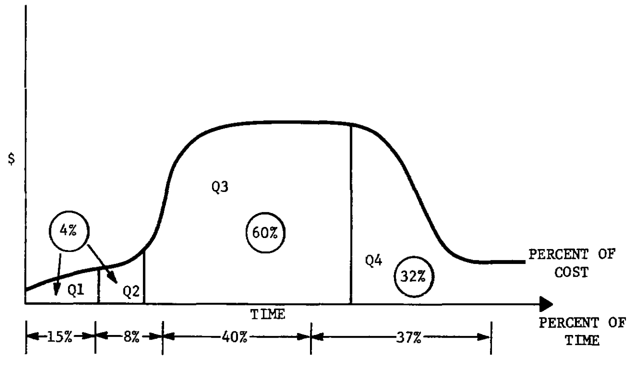


Figure 2—Idealized software life-cycle

The time axis

The percentages of time thought to be denoted to each phase of the software life cycle as implied by the shape of the idealized curve are as follows: Conceptual 15%, Requirements 8%, Development 40%, and Operations 37%. This differed from the reported experience of several large DoD projects.[33] In actual practice the conceptual phase accounted for 30% rather than 15%, while development took only 12% (compared to 40%). The requirement phase accounted for the same percentage of time in actual practice as was expected, while the operations phase (implementation and maintenance) lasted longer in actual practice (50%) than is implied by the curve (37%).

In absolute terms, these projects spanned 16 years from inception to termination. The percentages translate into a conceptual phase of 4½-5 years; a requirements phase of about 1.5 years (these two were actually performed simultaneously for about 6 months); 2 years for development and 8 years for operations. The requirements phase consists of the preparation of specifications, drafting an RFP and the evaluation and selection of a vendor. About half (3½% of Total Life Cycle Time) the time was devoted to specifications. The RFP's took slightly less (2½% TLCT) with about 4-5 months (2%) devoted to review, evaluation and selection. The components of the development phase (2 years) are more difficult to characterize by time, since the steps within are either overlapping (requirements analysis and design) or simultaneous (code, test, document).

Relative cost of software life cycle phases

The relative costs of each of the four phases of the software life cycle can also be inferred from the shape of curve presented in Figure 2. To verify these inferences, data from several studies are pieced together and a composite software life cycle (Figure 4) is constructed and presented in a following section. The shape is compared to the idealized versions found in the literature. It should be remembered that the purpose in developing this composite is to obtain estimates of the relative costs of each phase to use in the determination of the potential effects of instituting various forms of quality assurance. Therefore, the cost balance between development and maintenance as well as among steps in the development phase were of greatest concern.

Operations vs. development

The balance between development and operation depends primarily upon the length of the maintenance period used in the calculations. To standardize these calculations for comparison purposes a maintenance time period equal to 50% of the total life cycle (or 8 yrs) will be used. A study[9] which monitored costs fairly closely from requirements through *one year* of maintenance reported expenditures (in terms of man years) for Requirements, Development and Operations.

These figures were weighted (3 for management; 2 for programmer and 1 for staff support) to determine costs incurred. Assuming a negligible cost for the conceptual phase, say 1% and an operational life of 8 years, the percentage of total costs incurred by each of the four phases of the life cycle were calculated as follows: Requirements 1.5%, Development 51.3% and Operations 46.2%. The ratio of Development to Operations (Implementation and Maintenance) in this case would be 1 to 1.1.

Implementation is difficult to separate from development and maintenance since it in reality is a transitional period between the two. For this reason data about implementation is hard to find and interpret. This being the case the remainder of this paper treats operations as essentially equivalent to maintenance.

A look at cost data available for development vs. maintenance costs for OS releases 18, 19, and 20.0[13] are even more heavily weighted toward maintenance with ratios of 3 to 1 for OS 18 with only two years of maintenance included and 1.25 to 1 for all three releases with only one year of maintenance included. The experiences reported on in this section with respect to the balance between development and maintenance costs show that the costs of maintenance consistently exceed costs of development. Since QA would be expected to have the greatest impact upon costs in the operations phase, a conservative cost equation (conceptual cost+requirements cost+development cost=operations cost) will be used to *minimize* the estimated potential for QA.

Relative costs within the development phase

The activities undertaken during the development phase can be grouped into (1) analysis and design, (2) coding and debugging and (3) testing or validation.

The ratio of the cost of these activities to one another is often thought to be a function of the complexity of the system to be developed. That is, a non-linear (exponential) relationship is said to exist between complexity and the cost of testing. Testing costs are highly related to the number and severity of errors to be discovered and fixed, the number of which is related to system complexity. Proponents of QA will argue that this exponential relationship need not be the case if proper management (including a good QA plan) is exercised. Since the success of QA is directly related to error rates and error rates are the underlying causes of the cost relationships among the activities undertaken during development, this section will concentrate on the ratio of testing (or validation) to the total of development costs.

	Analysis and Design	*Coding and Debugging*	*Validation*
SAGE	39%	14%	47%
NTDS	30%	20%	50%
GEMINI	36%	17%	47%
SATURN V	32%	24%	44%
OS/360	33%	17%	50%
AVERAGE	34%	18%	48%

SOURCE: 14

Figure 3—Breakdown of development costs for selected systems

A study[8] which looked at the relative costs of design, coding and debugging in relationship to validation reported that the ratio of validation (testing) costs to the total development effort ran between $1/3$ to $1/2$.

Figure 3 gives a breakdown of the development phases of five large projects. The results[14] are very consistent from project to project and in the range of the results of the first study referenced. The range $1/3$-$1/2$ also includes the experience from ALPHA-6[9] reported on earlier in this chapter.

A composite software life cycle, based upon a 16 year length (50% operational life) and the relative costs for the four phases given in Figure 3, is presented in Figure 4. The shape is far more leptokurtic than the

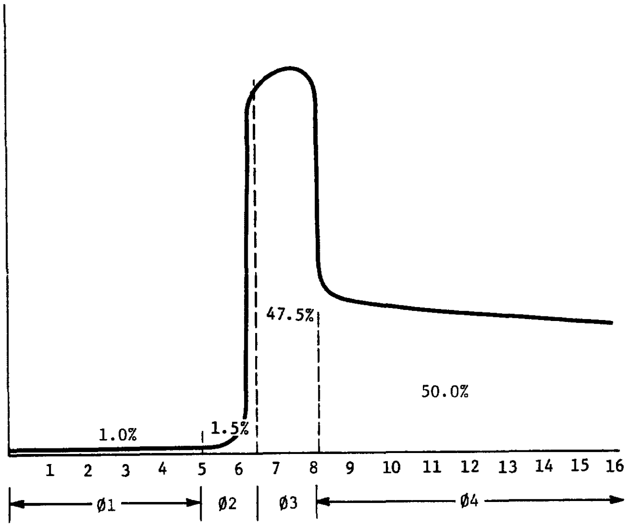


Figure 4—Composite software life-cycle

"idealized" curves found in the literature with the length of the maintenance tail spreading its significant costs over many years. This is due in larger part to a contraction of the development phase. The visual impact of the meager resources applied to the Requirement phase also represents a departure from the idealized shape.

THE COST OF ERROR

A measure of system quality is the number of errors which occur. Hence, ratios of one kind of error to another have been proposed[22] as indicators of quality software. It is taken on faith that well designed systems can be put together with little resultant error, and for those errors which occur, the mean age of the errors becomes a vital statistic with which to judge software.

This section is devoted to estimating the source, kind, type and severity of errors generated during development. It would be of interest to examine the requirements stage to place a value on the "errors" which originate there and trace their impact throughout the rest of the life cycle; but aside from intuitive feelings about their impact no real data appears to be available.

Frequency and severity of errors

No two researchers group errors in quite the same way. As a result, the available information on software errors had to be interpreted and classified based upon the explanations provided in individual studies. Errors are classified in this paper as either design, logic or syntax. These categories are sufficient for the purpose at hand. Design errors are those which require changes in the specifications used by the programmers. Usually they represent a lack of understanding (or proper communication) of a computation or process, which results in the wrong "problem" being solved. Logic errors occur when the system design is translated into programmable form (detailed flow charts). Syntax errors are self-explanatory. Few of the studies of software errors present actual data pertaining to frequency and severity. Taken together[1,4,6,8,47] those that present some data all report design error as occurring most frequently. Ranging from a high of 64%[1] to 46%.[6] Syntax errors were reported to be about 15% of the known errors. Logic errors ranged from 21% to 38%. The significant point to note is the large percentage of design errors.

Available data on cost of detection and correction reveals that design errors cost the most to diagnose and fix. Syntax errors are reported to be more of a nuisance than a significant cost particularly with the use of automated precompiler processing.

Origin and detection of errors

Where errors originate as well as when and how they are discovered are important inputs to the design of an effective QA plan. Syntax errors originate, surface and are resolved within a brief period of time and for all intents and purposes can be considered totally encompassed within the process of coding. Such is not the case with design and logic errors. Design errors can be caught during a design review (if there is one), during preparation of detailed flow charts or occasionally during coding. Simple logic errors (process before read) can be caught at compilation time or during program testing. Because of the numerous paths in any program which can be tested many logic errors are not observed until the validation, implementation or maintenance stages. A study of a large software development effort[1] found that 54% of the errors were not caught until acceptance testing or presumably until after development was complete. To make matters worse, the overwhelming proportion of these were design errors. Reported figures indicated that 70% of the design errors were not caught at earlier stages while by contrast 80% of the programming or logic errors were caught during development. If the mean age of error were calculated for this case it would be quite high due to the high percentage of design errors involved.

Estimation of the costs of errors

Using the three categories of error (design, logic and syntax) it appears that design errors account for at least 80% of the *total cost of error*. This percentage is arrived at by noting that about ⅔ of all errors caught are design errors; with logic and syntax errors making up about equal proportions of the remaining ⅓. Compared to the cost of tracking down and correcting coding errors, the cost of syntax mishaps is small. However, the cost of design errors is more than double (2½ times) that of coding error. The calculation of the contribution of design error to the total cost of error consists of taking the weighted (expected) cost of an error [% design errors × 2½ + % coding + syntax errors × 1] and dividing it into the contribution of design [% design × 2½] using the percentage given above 83⅓% of the total cost of error can be attributed to design errors. This relatively large contribution to total error cost should play an important role in the design of a QA plan and will be used as an input in the calculation of the potential effectiveness of quality assurance.

The next calculation which is required for the assessment of the potential of quality assurance is the percentage of total life cycle cost which can be attributed to error. Once this percentage is obtained, an estimate of the benefits of a QA plan can be developed based on a "tool by tool" analysis of the kind of error it addresses (design, logic or syntax) and the percent error reduction claimed or experienced. These calculated benefits can then be compared to the cost associated with these components of a QA plan for a final assessment of the economics of quality assurance.

To estimate the cost of error the following method was used. For a slightly conservative estimate, it was assumed that (i) all costs in design (ii) coding and (iii) documentation to be non-error related. All checkout and validation costs (recognizing that only some of these costs can be reduced by reducing the number of errors, since some costs are fixed) were attributed to error. From the ALPHA-6 data then 47% of development cost (assuming the code, test, and document costs were equal) could be traced to errors. Further data on developments costs for several large systems (given in Figure 3) averaged almost exactly the same percentage (48%).

Maintenance costs can be attributed to correcting errors and to enhancements, but "enhancements" often result from initial design errors. For the sake of discussion assume that half can be directly related to error. This amounts to the conservative estimate of almost half of the total life cycle costs (47.6%) being directly tied to error (see Figure 5). On the cost basis of a large system, the total cost of error is in the hundreds of millions. If quality assurance methods can reduce error by even small amounts, they would appear to be worthy of serious consideration. For example, a 10% reduction in error (⅔ Design, ⅓ logic and syntax) as they have been reported in the studies reviewed would represent a saving of almost $25 million based upon a relatively large effort (cost= $½ billion over the 16 year cycle). A five percentage error reduction (only ½ design) would result in a savings of over $10 million.

THE EFFECTIVENESS OF THE TECHNIQUES AND TOOLS OF QUALITY ASSURANCE

The assurance of quality can be brought about by any number of different approaches which have been suggested. These range from essentially project management techniques to methodologies of design to syntax checking tools. Many of the methods which will be discussed in this section can be expected to have a much broader impact on design, development and implementation than is pertinent to a discussion of quality assurance. This section will address the impact these methods have on error rate and error-related productivity.

Error Type	*% Total Errors*	*Relative Severity*	*% Total Cost of Error*
Design	2/3	2.5	83+%
Logic	1/6	1.0	8+%
Syntax	1/6	1.0	8+%
	Development Phase	*Operations Phase*	*Both*
% Total Life Cycle Cost	47.5%	50%	97.5%
% Cost Due to Error	48%	50%	—
% Total Life Cycle Cost Attributed to Error	22.6%	25%	47.6%

Figure 5—Error and software life cycle costs

In some cases, their contribution to quality is rather straightforward, particularly for error detection tools. However, for those which promise the most sweeping reforms, essentially those dealing with management or design effectiveness, measurement is difficult and little concrete information is available.

It is the purpose of this section to analyze based upon available data the potential of quality assurance in terms of the cost of error, development productivity and the cost of quality assurance. In the following paragraphs, some of the most widely discussed techniques and tools will be reviewed.

Structured programming

The advantages touted for Structured Programs range from improved program design to improved documentation. Improved design is linked to fewer design errors and fewer logic/programming errors. Fewer statement types are linked to fewer syntax errors and an almost self-documenting program. Fewer errors imply greater productivity during development and reduced operations costs. Further, the streamlined design is claimed to be easier to upgrade and enhance. Finally, the planning and conceptualization required by Structured Programming is said to enhance the performance of project management.

Reported increases in error free productivity ranged from 50%[25] to 125%[47] with the introduction of Structured Programming while error reductions of between 30%-90% were reported by another study.[17] Quantitative results of ease of enhancements were not found, however, a study of the development cycle[56] estimated 25% reduction in the elapsed time from requirements to implementation, from 6 years to 4.5 years.

Top-down development

The essence of Top-Down Development is simultaneous systems integration and development which results in a viable, executable, if rather skeletal system at an early state. This development approach amounts to an ordering of the sequence of system decomposition decisions beginning with a simple description of the entire system or process and continuing with successive refinements until a programmable design is reached. Top-Down Development is a natural companion of Structured Programming, so much so that the two concepts are often confused. The claimed advantages of this approach include the increased ease of implementing a QA plan with a resultant reduction in design error and productivity improvements associated with the systems integration and testing efforts during the development phase.

Perhaps the most significant advantage claimed from a QA perspective, is the early existence of a complete system's design replete with the design specification of system components and interfaces. Not only does such a document enhance the changes of a coherent and consistent design, but it also serves as a vehicle for establishing a correspondence with "user" oriented functional specs. The system components are placed into perspective for all to see and comment upon. Misunderstandings that often were not surfaced until acceptance testing can be resolved at this time. Design problems often not found until systems integration may be corrected reducing the high cost currently associated with these problems.

The incorporation of the testing function throughout development, made possible by the continual existence of a testable system, offers QA with an opportunity to be more of a pro-active force in development.

There are recognized pitfalls as well. Care must be taken to ensure design feasibility in terms of existing software and hardware, since actual coding is significantly delayed.

Holistic design is difficult to achieve and false starts are likely. However, when weighted against the known shortcomings of bottom-up design there is little question that a Top Down approach when combined with some common sense offers substantial advantages to both developer and user.

Hard estimates of the reduction in error and increases in productivity from the use of this approach alone are not readily available. However, when used in conjunction with Structural Programming and a Development Support Library,[25] a productivity improvement of over 300% (when compared to a system using a Development Support Library alone) was experienced. With Structural Programming alone, productivity gains of 50%-100% were experienced; thus, the addition of a Top Down Design approach seems to further enhance performance significantly.

For the purposes of this analysis, the expected performance of this approach will be conservatively bounded from above. In terms of development productivity, a very conservative range which includes gains made by reduced systems integration and testing, and by better manpower and computer time scheduling would be between a 5-10% improvement in productivity. This improvement could be reasonably expected from the savings in the integration step alone.

As far as design errors are concerned, the increased attention to overall design could be expected to reduce configuration and architecture errors significantly and virtually eliminate errors in the specification of offered system functions. One study[6] showed that machine configuration and architecture errors accounted for just over 20% of all design errors while errors in the functions offered accounted for about 25% of the design erorrs. Both are susceptible to being caught early. An examination of specs by others not involved in their formulation resulted in the detection of between 30%-40% of these errors. An increase from this to a 50% rate of error detection might realistically be achieved by the use of Top Down Design.

Other recent innovations

In addition to Structured Programming and Top Down Development, a number of other approaches to improving software quality and productivity have been advanced. Among these are the techniques of the Chief Programmer Teams, Egoless Programming, and automatic or semi-automatic tools ranging from Design Assertion Consistency Checkers to Automated Test Case Generators.

The management oriented techniques are aimed at achieving increased communications and coordination while the automated tools seek to provide complete, systematic and low cost verification. This section will briefly explain some of these innovations concentrating on the contribution or impact likely on the performance of the QA functions.

Programming organizations

In this section, the effects of the Chief Programmer Team, Egoless Programming and Democratic Team Organization on the performance of the QA function will be addressed. Egoless Programming and Democratic Teams are essentially loosely structured programming environments in direct contrast to the Chief Programmer approach which is highly structured. It is interesting that the changes from current practice being advanced to improve software quality are in opposite directions. Both approaches, however, take aim at the individualist who becomes ego-involved with code to the extent that error detection is thwarted. The loosely structured approaches attack this problem directly by eliminating "ownership" of code to reduce defensiveness. The Chief Programmer Team approach is meant to be employed in conjunction with Structured Programming and Top Down Design which systematically eliminates tricks and gimmicks in programming and imposes ridged forms. Users of both types of approaches claim better communication leading to reduced misunderstandings and error rates. On the one hand, the Chief Programmer Team approach is criticized for being too authoritarian while the other approaches are said to tend to alleviate the individualist and require more sophisticated management techniques. Experience indicates that managing bright

	Methods of Detection			
Error Type	*Manual Inspection*	*Formal Methods (Simulation, etc.)*	*Tests Runs*	
Design	45%	20%	35%	100%
Logic/ Coding	24%	22%	54%	100%

Figure 6—Error detection for design and logic/coding error types

creative staff is no mean task regardless of the techniques employed. The key seems to be in the actions taken to increase the understanding and clarity of assignments not in what abstract management philosophy is employed.

Automated tools for quality asssurance

The literature contains countless tools developed to check out design, flow charts, code and even documentation systematically and quickly. Their performance can more easily be measured than the techniques previously discussed, but their contribution to the potential of an overall QA plan is limited. Their very nature (highly specified and deterministic) limits their effectiveness in dealing with other than highly structured situations. Thus, these tools are most applicable to the detection of errors in code and simple sorts of logic errors rather than major flaws in program logic or design approach. Nevertheless, they can significantly contribute to increased productivity, earlier detection and hence some reduction of the "ripple" effect (19% of the errors introduced as a result of error correction[55]). An analysis of error types and means of detection[6] showed that (See Figure 6) manual inspection uncovered only 24% of logic and coding errors compared to 45% of design errors indicating the potential for the use of automated tools. Such tools could have an impact in reducing the percentage of logic and coding errors (54%) not caught until testing. One study gave evidence to support this feeling[21]. The use of automated instruction and path checkers (ASSIST and NODAL) reportedly catch between 67%-100% of the errors and at between 2-5 months earlier than they would have otherwise been detected. Automated error checking is currently at the state of development where it is either language or application specific and it would probably be of marginal value to develop such a tool for a specific project.

Figure 7 summarizes the results with respect to the reported effectiveness of quality assurance methods and shows the dollar impact that improvements in development productivity can have based upon a project whose total life cycle costs equal $.5 billion.

SUMMARY AND CONCLUSIONS

This section places the relevant estimates developed during this report in perspective and highlights important aspects in the assessment of the economics of Software Quality Assurance. This paper first addressed the software life cycle to identify the areas which could be improved by a QA plan. Second, an examination of the frequency of software error, its sources or origins, methods of detection and associated costs was presented. This was followed by an examination of some of the methods and techniques suggested for quality assurance. Highlights of these examinations and analyses follow.

Summary of findings

The examination of the software life cycle revealed that costs were concentrated in the Development and Operations phases. The typical Development Phase accounted for just under 50% of the total costs while lasting about 2 years (12% of a 16 yr. cycle). About half of the development costs were spent on check-out and testing activities in contrast to about $1/3$ for analysis and design and $1/6$ for actual coding. The Operations phase while consuming just under 50% of the total life cycle costs was spread over an eight year period.

Errors were classified into three types (design, programming/logic, syntax). The last accounting for some 15% of all errors. Design errors outpaced program/logic errors by a little less than 2 to 1 accounting for a little more than half of all errors. Program/logic errors ran about one-third of the total.

The severity of errors, as measured by the cost of detection and correction, was found to be higher for

Technique	*Error Reduction*	*Productivity*		*\$ Impact of 1% Improvement In Development*	*Potential Impact*
		Range	*Mid-Point*		
Structured Programming	30-90%	50%-100+%	75%		$175 Million
Top-Down Design	Substantial	10%-200%	100%		$250 Million
Management Organization	—	—	—	$2,375,000	—
Automated Tools	Caught earlier	Up to 25%	10%		$ 25 Million

Figure 7—Performance of quality assurance techniques

design errors than program/logic or syntax errors (least costly). Weighted by costs it was calculated that design errors accounted for just over 80% of the total cost of error. In terms of the total software life cycle then, with 47.5% of its costs in development and 50% in maintenance, the cost of error could easily run over 50% of the total software life cycle cost.

To combat error and improve software quality a variety of methods have been suggested. Preliminary reports have been encouraging in both the areas of productivity improvement and error reduction.

Conclusions

While the data drawn upon comes from a large variety of sources (different systems, different environments and from studies using different definitions and analysis methodologies), the experiences reported were so compatible that, while more detailed data is necessary for the actual development of a QA plan specific to a given set of system and organizational circumstances, the conclusion that QA can be cost effective is inescapable.

From the analysis presented in this paper, the development of a QA plan should concentrate on techniques and methods for the early detection and elimination of design errors. The researchers reporting on the development of ALPHA-6[9] indicated that if more resources were applied during design, it would have resulted in substantial savings in the costs of testing and maintenance. An extrapolation of the data they presented gives a multiplicative factor of 5; that is, a dollar more spent in design would have saved 5 dollars spent on testing and maintenance. While this example may be unusual, it, together with the fact that a significant portion of total system cost can be attributed to error point to the cost impact that a QA function can provide.

A parameterization of the impact that error and productivity improvements have on total software system costs based upon a $½ billion total life cycle cost (about $250 million for S/W Development) has been made. For each 1% of error reduction (½ coding + ½ design) a savings of just over $1½ million could be expected. For each 1% improvement in Development productivity a saving of $2,375,000 could be expected. It should be noted that Design errors have more than double the impact than do coding errors.

Thus the leverage of QA in large programs is significantly high to warrant serious consideration. The costs of developing and implementing a QA plan are difficult to specify for a given organization, especially in light of their management considerations. However, even with the additional expense QA still promises to be cost-effective. For example, if management overhead for software development is approximately 5% of development costs and a QA plan increased this overhead by ¼, then a reduction of error by approximately 1% (coding) alone could offset these additional costs.

REFERENCES

1. Boehm, B. W., et al, "Some Experince With Automated Aids to the Design of Large-Scale Reliable Software," *IEEE Transactions on S/W*, TRW, March 1975.
2. Boehm, B. W., "Software and Its Impact—A Quantitative Assessment," *Datamation*, TRW, May 1973.
3. Brown, J. R., et al, "Evaluating the Effectiveness of Software Verification—Practical Experience With an Automated Tool," *AFIPS Fall Joint Computer Conference*, December 1972.
4. Shooman, M. L., et al, "Types, Distribution, and Test and Correction Times for Programming Errors," *IEEE Transactions*, Bell Labs, March 1975.
5. Schneidewind, N. F., "Analysis of Error Processes in Computer Software," *IEEE Transactions*, Naval Postgraduate School, March 1975.
6. Endres, A., "An Analysis of Errors and Their Causes in System Programs," *IEEE Transactions*, IBM Lab, Germany, March 1975.
7. Rubey, R. J., et al, *Comparative Evaluation of PL/1*, USAF ESD-TR-68-150, April 1968.
8. Rubey, R. J., "Quantitative Aspects of Software Validation," *IEEE Transactions*, LOGICON, March 1975.
9. Buda, A. O., et al, "Implementation of the ALPHA-6 Programming System, IEEE Transactions," *Academy of Sciences* USSR, March 1975.
10. Ramamoorthy, C. V., et al, "Testing Large Software Evaluation Systems," *IEEE Transactions*, CSD, ERL, University of California, Berkeley, March 1975.
11. Alexander, T., "Computers Can't Solve Everything," *Fortune*, May 1969.
12. Boehm, B. W., "System Design," *Planning Community Information Utilities* (ED) H. Sackman, AFIPS Press, 1972.
13. Barry, B., et al, *Software Life Cycle Considerations*, IBM, January 1974.
14. Boehm, B. W., "Some Information Processing Implications of Air Force Space Missions: 1970-1980," *Astronautics and Aeronautics*, January 1971.
15. Vick, C. R., *Specification for Reliable Software*, EASCON, 1974.
16. Brown, J. R., et al, "Evaluating the Effectiveness of Software Verification—Practical Experience with an Automated Tool," *AFIPS Conf.*, 1972.
17. Cammack, W. B., et al, *Improving the Programming Process*, IBM SDD TR002483, October 1973.
18. Cheng, L. and J. E. Sullivan, *Case Studies in Software Design*, MTR-2874, Volume I, June 1974.
19. Boden, W. H., "Designing for LCC," *EASCON 74*, pp. 624-29.
20. Knight, C. R., "Warranties as a Life-Cycle-Cost Management Tool," *EASCON 74*, pp. 621-623.
21. Mangold, E. R., "Software Error Analysis and Software Policy Implications," *EASCON 74*, pp. 123-27.
22. Mills, H. D., "How to Buy Quality Software," *EASCON 74*, pp. 120-22.
23. Nashman, A. E., "Software Development Management: The Key to Quality Software Products," *EASON 74*, pp. 31-35.
24. Oliver, P., "Observations on Software Reliability," *EASCON 74*, pp. 126-29.
25. Baker, F. T., "Structured Programming in a Production Programming Environment," *IEEE Transactions on S/W Rel.* 75, pp. 172-185.
26. Rain, M., *Two Unusual Methods for Debugging S/W Software Practice and Experience* 3, pp. 61-63.

27. Katzenelson, J., *Documentation and Management of Software Project*, SP & E 1, 2, pp. 147-157.
28. Peadi, P., *Quality Control for Computer Programming: A Final Report on an Initial Study*, SDC, Santa Monica, California, September 1965.
29. ———, *QC for Systems and Programming, A Survey of the Literature*, SDC, March 1965.
30. Connolly, J. T., *Software Acquisition Management Guidebook: Regulation, Specifications and Standards*, MTR-3080, The MITRE Corporation, June 1975.
31. Clapp, J. A., *Major Contributions to Software Engineering in the 1980's*, MTR-2791, The MITRE Corporation, January 1974.
32. ———, *A Software Error Classification Methodology*, MTR-2648, Volume VII, The MITRE Corporation, June 1973.
33. Reifer, D. J., "Automated Aids for Reliable Software," *IEEE Transactions on S/W Reliability*, March 1975, pp. 131-42.
34. Wulf, W. A., "Reliable Hardware-Software Architecture," *IEEE Transactions on Software Reliability*, March 1975, pp. 122-30.
35. Cicu, A., et al, "Organizing Tests During Software Evolution," *IEEE Transaction on Software Reliability*, March 1975, pp. 43-50.
36. Williams, R. D., "Managing the Development of Reliable Software," *IEEE Transactions on Software Reliability*, March 1975, pp. 43-50.
36. Williams, R. D., "Managing the Development of Reliable Software," *IEEE Transactions on Software Reliability*, March 1975, pp. 3-8.
37. Ceoff, N. S., "Development Project Costs," *Journal of Systems Management*, September 1974, pp. 14-17.
38. Aron, J. D., *Characteristics of the Program System Development Life-Cycle*, IBM FSD 74-0180.
39. Pietrasanta, A. M., "Resource Analysis of Computer Program System Development," *On The Management of Computer Programming*, G. F. Weinwurm, (Editor): Auerbach Publishers, 1970.
40. Brooks, F. P., Jr., "Why is Software Late," *Data Management*, Volume 9/8, August 1971.
41. Clapp, J. A. and J. E. Sullivan, *SIMON: Finding the Answers to Software Development Problems*, MTP-159, The MITRE Corporation, May 1974.
42. Cheng, L. L., *Some Case Studies in Structured Programming*, MTR-2648, Volume VI, The MITRE Corporation, June 1973.
43. Fleischer, R. J., *Effects of Management Philosophy on Software Production*, MTR-2648, Volume II, The MITRE Corporation, June 1973.
44. Corrigan, A. E., *Results of an Experiment in the Application of Software Quality Principles*, MTR-2874, Volume III, The MITRE Corporation, June 1974.
45. Schiff, J. D., "An Overview of the Software Life-Cycle Process," *Proceedings of the Aeronautical System Software Workshop*, Dayton, Ohio, April 1974, p. 108.
46. Prywes, N. S., *Research on Automatic Program Generation*, Report 74-05 University of Pennsylvania Moore School of Electrical Engineering, January 1974.
47. Boles, S. J. and J. D. Gould, *A Behavioral Analysis of Programming—On the Frequency of Syntactical Errors*, IBM RC 3907, June 1972.
48. McGonagle, J. D., *A Study of a Software Development Project*, J. P. Anderson and Company, September 1971.
49. Nichols, B. S., *Practical Experience With Structured Programming*, Bell Systems, November 1973.
50. Mill, H. D., "Top Down Programming in Large System," *Debugging Techniques in Large Systems*, R. Rustin (ED) Prentice Hall, 1970.
51. Baker, F. T., "Chief Programmer Team Management of Production Programming," *IBM Systems Journal* 11, 1972.
52. Sackman, A., *Man-Computer Problem Solving*, Auerbach Publishers, 1970.
53. Weinberg, G. M., *The Psychology of Computer Programming*, Van Nostrand Reinhold, New York, 1971.
54. Baker, F. T., "System Quality Through Structured Programming," *AFIPS Conference Proceedings*, 1972, pp. 339-343.
55. McGonagle, J. D., *A Study of a Software Development Project*, James P. Anderson and Company, Los Angeles, California, 1971.
56. Haile, A., *Command and Control Information Processing in the 1980's* (USAF-CCIP-85) Presentation in DoD Computer Institute Seminar IX, November 1972.
57. Asch, A., et al, *DoD Weapon Systems Software Acquisition and Management Study*, Vols. I and II, The MITRE Corporation, MTR-3908, 1975.

Implementation of quality control in software development

by FRANK TSUI and LEW PRIVEN
IBM
Gaithersburg, Maryland

ABSTRACT

In recent years, many tools have been developed in the areas of programming techniques, design methodology, development processes, and test strategies. All of these are aimed at producing a quality product with a minimum cost. Together with emergence of these tools, there is a recognition of the problem of how to effectively implement them into a total systems development environment. One of the factors to a successful development is the ability to exercise proper control over the various development processes.

We will first establish the objectives for software quality control. Then, descriptions of the terminologies defining the various development processes will be presented in order to provide a common background. A basic philosophy of the entire development procedure and the notion of *Continuous Integration* will be presented. The introduction of quality control into this procedure will be discussed. Finally, some justification for such an approach will be shown.

INTRODUCTION

A large amount of effort has been spent on developing programming and design techniques. The various tools and methodologies[1,2,4,5,12,15] are aimed at different steps in the development processes. Basically, these tools may be classified into the following categories in terms of their intended applications:

(i) Design, Code and Documentation
(ii) Test and Verification
(iii) Control of the Development Processes

In order to attain a good quality product, it is essential to understand how and where these tools apply in the development activities. The subject of this paper comes under the third category, Control of the Development Processes.

In this paper we will present a unifying paradigm of the entire development procedure. Within this global system, several development cycles for different functions may be in progress simultaneously. The development cycle is similar for each function. That is, the activities within each development cycle are fairly standard and flow in a fixed sequence much like an assembly line. For example, if we examine how a programmer does his job, normally the following activities take place:

(i) Design/Document
(ii) Code
(iii) Test
(iv) Integrate
(v) Performance Measure
(vi) Release

This is a "natural" process; id est, if left alone, most programmers would follow this sequence in implementing a program, except, possibly for the documentation part. Since programming lends itself to a natural flow, the process or the development cycle should keep this intact.

In lieu of the yet experimental and basically primitive automatic program correctness verifiers,[11] quality control is obtained via the introduction of inspections and reviews[7,8] into the development cycle. Then two additional steps called Error Prone Analysis and Post Functional Test Analysis will be introduced to monitor the development cycle and the entire development procedure.

GOAL OF QUALITY CONTROL

Today's environment demands that programming products be of "high" quality. From the users' point of view this means: functionally complete, easily usable and installable, and low error rates. Additionally, to the developer it means that the schedule must be met at the lowest cost. This is certainly a tall order based on the past performance of the programming community, but it can be achieved with the proper management and quality control techniques.

The main objective here is to ensure that only a minimal number of software problems exist. This may not be as obvious as it appears at first glance, for it is not sufficient that just the final product contains a small amount of problems. It has been the experience[4,7] that correcting a problem in the final product will require much more effort than correcting that same problem in the early phase of the development. Thus it is essential that errors be detected and corrected as early as possible and only a minimal amount of problems be allowed to slip from one phase of the development to the next.

The secondary objective, although it may be just as important as the main objective, is to increase the ease of usage and future maintenance of a product. As the cost of producing a completely new system increases, the property of being able to enhance and build upon the existing system becomes significant.

TERMINOLOGIES AND DEFINITIONS

In this section some basic notions of Design, Code and Test will be explicitly defined in order to facilitate the introduction of the concept of inspection or review.

The Design phase of the development cycle is composed of two levels: High-Level Design and Detailed Design. The High-Level Design contains the following information.

(A) An over-all functional specification describing *what* is to be performed.

(i) For each major function, the modules' names should be specified; and within each module, the description is carried to the level where each statement represents approximately 20 lines of executable source code.
(ii) All control blocks with major fields and their purposes identified.
(iii) For each function, all the macros that need to be designed should be listed and described to the depth where each statement represents approximately 20 lines of executable code.
(iv) For each module, all data files and tables should be presented along with a brief description of their purpose and method of access.

(B) The dependencies for each major function are presented by modules, identifying both internal and external cases.

(i) Internal dependencies specify those macros and control blocks which are shared by several modules.
(ii) External dependencies indicate all the modules, control blocks, and macros needed but were not included in the overall functional specifications described above.

(C) The linkages between functions (by modules) are specified.

(i) Parameter list is identified.
(ii) All system Generation/Initialization requirements must be described.
(iii) All locks and authorization into data files are described.

(D) For each function, a control flow (in some flowchart form—preferrably with a predetermined Structured flow chart) describing the logic should be given by modules.

The Detailed Design is a refinement of the High-Level Design. Thus the information described above for High-Level Design serves as the input to the Detailed Design phase. The information contained in the Detailed Design is used as the input to coding. It contains the following:

(A) A complete "Structured" logic flow (by modules) should be given.

(i) Each box in the flowchart should represent approximately 20 lines of executable source
(ii) Reference to control blocks and data files should be by field names.
(iii) Macro invocations should specify all required parameters (by values).
(iv) Each predicate in the flowchart should have a corresponding "purpose" description.

(B) Data structure should be clearly specified. Each data file and table should be described down to each field identifying the following:

(i) intended usage
(ii) attributes
(iii) access method
(iv) value range (if any)

The coding phase utilizes the Detailed Design as its input. Here the Detailed Design is converted to source code and machine compiled or assembled. Each module should be at least compilation or assembly error free. There should be a module prologue preceding the actual code, stating the following about that module.

(a) major functions
(b) control blocks referenced (if any)
(c) macros used (if any)
(d) Entry-Exit conditions (if non-standard)
(e) linkages (if any)
(f) I/O parameters (if any)
(g) list of logic revisions made since last version (if any)
(h) version number and date
(i) module owner

An important point here is that the code must follow the Detailed Design. Structured flowchart with its corresponding Structured Programming Language would make this process much easier than otherwise.

The Testing phase depends heavily upon the documents generated during High-Level and Detailed Design. Generally, there are three categories of testing:

(a) Functional Test
(b) Component Test
(c) System Test

The functional test refers to the verification of each function utilizing a series of test cases within the intended environment. This may be the most time consuming test, depending on how detailed the test cases are. The component test involves the verification of inter-functional activities. The system test is performed to insure that no part of the system is ill-affected (including performance) and that the system as a whole does not crash even under unusual, non-intended environment. An extensive description of testing processes may be found in Reference 6.

IMPLEMENTATION OF QUALITY CONTROL

Before we can meaningfully discuss Quality Control, there has to be an underlying development philosophy. In this section, we will first present a paradigm of the complete development process. Within that paradigm the development cycle with the built-in controls will be discussed.

In order to insure low error rates, some quality control techniques must be utilized. This implies a defined procedure that can be instrumented to provide a) the necessary points for measurement and b) feedback path to correct defects. As stated earlier, the "natural" sequence of activities of Design, Code, Test, Integrate, Performance Measure, and Release should be kept as part of the process. Together with these activities, some checkpoint must be obtained before an activity is considered complete and another begun. The question is whether there is any one activity that plays an overriding role in defining the interaction between the other activities. Design seems to be the logical choice, but integration will be shown to be the key in defining the sequence of design and development activity.

Similar to building a house, the sequence that programs are put together is very important. In the case of a house, first the foundation, then the outside walls, then the roof, etc. are "put together" in that sequence. The parts may be built independently and simultaneously. Once the "basic" framework of the house is completed, the other components such as kitchen cabinets, toilets, etc. may be added separately. The same concept holds true for programs; there is a main functional path that must be completed first. Then the other functions can be added until the entire product is complete. This technique works best when functions are added individually. The technique is called Continuous Integration; it provides two significant attributes to the programming, development process. This is conceptually similar to Iterative Enhancement.[3] First, by adding functions individually to an already working base, error isolation can generally be narrowed down to the newly added function or interface areas. This is critical as the system becomes more complex. Second, the development of each discrete function entails the same development cycle and can be tracked through the process by a standard set of quality control techniques.

The concept of Continuous Integration can be easily demonstrated as follows:

		Modules			
		A	B	C	D
New Functions	1	X	X	X	X
	2		X		X
	3	X	X	X	

Figure 1

Modules A, B, C and D are already developed, and they form the working base. Now, three new functions need to be added.

Figure 1 shows the relationship between additional, new functions and the impact to the modules. The crosses represent "affected." With Continuous Integration the three functions would be developed independently; although they can be developed simultaneously. As any one function is completed, it is integrated into the working base and a new base is formed.

Consider the alternative. Had we developed them by modules, the delay of any one module would affect the completion of at least two functions. If module B is delayed, nothing will be completed. This can be disastrous. The testing and verification of functions are also difficult with the Modular approach. All affected modules must be completed at the same time before any one function can be tested. Suppose modules A and C were completed first. Functional verification can not be performed until the most "affected" module B is completed with all its new enhancement. If the modules were developed at physically different sites, then the problems are even more complex.

Continuous Integration utilizes a "blueprint" for putting the system together called the system build plan. This plan is used to determine in what sequence code must be developed and, thereby, the design sequence. Figure 2 represents a unifying paradigm within which Continuous Integration is applied.

To have the processes started, assuming that the product has been defined via the commitment process, a system build plan must be produced. As shown in Figure 3, the production of the build plan is an iterative process, requiring trade offs in resource, function, and schedule. Once this has been agreed to for the

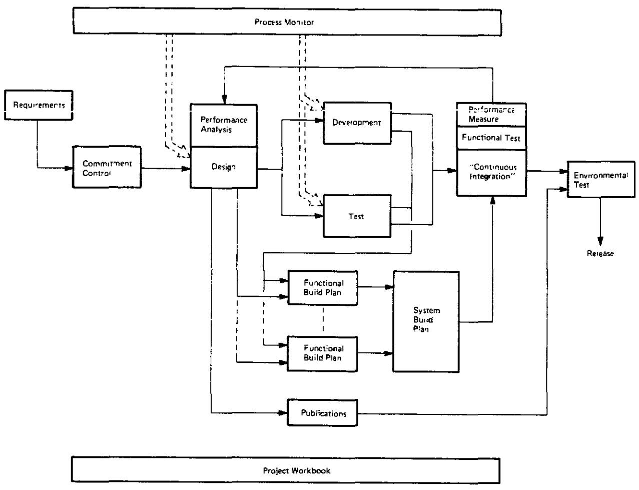


Figure 2—Programming processes paradigm

initial product definition, the processes shown in Figure 2 can be used to control the program production and introduction of changes.

Another area that is significant to the total development and quality control is documentation. A project "workbook" that contains at least the design documentation for each function must be maintained in an up-to-date manner. This documentation is used as:

(1) Source for user publication
(2) Source for test case development
(3) Control Information for Continuous Integration
(4) Source for quality control activities.

It is important to remember that Figure 2 is not a time sequence chart for the whole system; however, it is for any one function. At any given point in time, some activity will be taking place in each area. The processes outlined have many other facets that will not be discussed in this paper, for they are not relevant to the main topic of quality control. Refer to Reference 14 for further details.

In summary, the main features of the paradigm that are significant to quality control are:

(a) Process definition that provides for discrete turnover points between activities (e.g., design to code).
(b) Control over the input to the system via commitment control.
(c) A blueprint for putting the product together (e.g., system build plan). The system build plan is also the mechanism used to review the status of each function as well as the integrated system. Using this mechanism, inputs from the quality control activities can be used to modify the blueprint and/or reallocate resources to assist areas in need.
(d) Work units (functions to be developed) that move through the processes in trackable units.
(e) Documentation of the design which provides independent source material for quality control checking.

Now, we will show precisely how items (a), (d) and (e) mentioned above can be achieved by inserting quality control checkpoints into the development cycle of any major function. Recall that several functions may be developed simultaneously but they all follow a similar development cycle.

First a set of inspections or reviews[7,8] will be included into the cycle in order to attain the main objective of quality control, which was mentioned earlier. That is, with the inspections in place, the number of errors passed from one phase to the next will be greatly curtailed. Strict Exit Criteria will be imposed at each step such that the ensuing step may not start until the previous step is completed. In addition to the built-in controls of inspections and exit criteria, there should be explicit control steps in the development cycle. Namely, two new steps (a) Error-Prone Analysis and

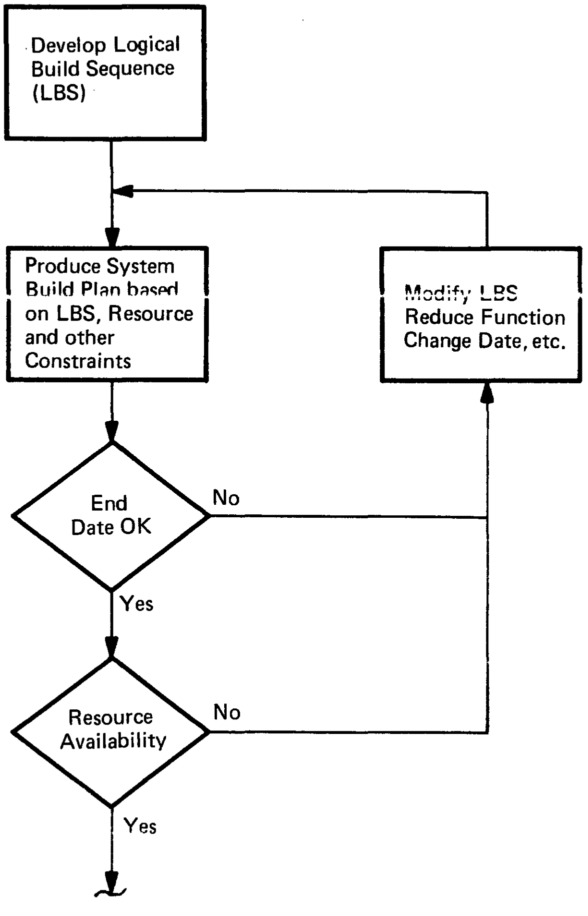

Develop Logical Build Sequence (LBS)
Produce System Build Plan based on LBS, Resource and other Constraints
Modify LBS Reduce Function Change Date, etc.
End Date OK
No
Yes
Resource Availability
No
Yes

Figure 3—System build planning

(b) Post Functional Test Analysis are included. There will also be a Test Planning process in the development cycle. Finally, we have the complete picture of a Development Cycle in Figure 4. All the terminologies (e.g., High-Level Design) follow the definitions provided in the previous section.

Each inspection involves the following 4 steps:

(a) Preparation
(b) Inspection itself
(c) Rework
(d) Follow-up

In the preparation step, all the appropriate and required outputs for that phase (e.g., Detailed Design) must be distributed to the inspectors. The inspectors are then given enough time to digest the material before the actual inspection. Typically, the inspectors are composed of a moderator, the designer of the function, the coder of the function, a testing representative, and a publications or documentation representative. Refer to Reference 7 for details on inspections. The objective of an inspection is to identify problems as opposed to offering solutions and better alternatives.

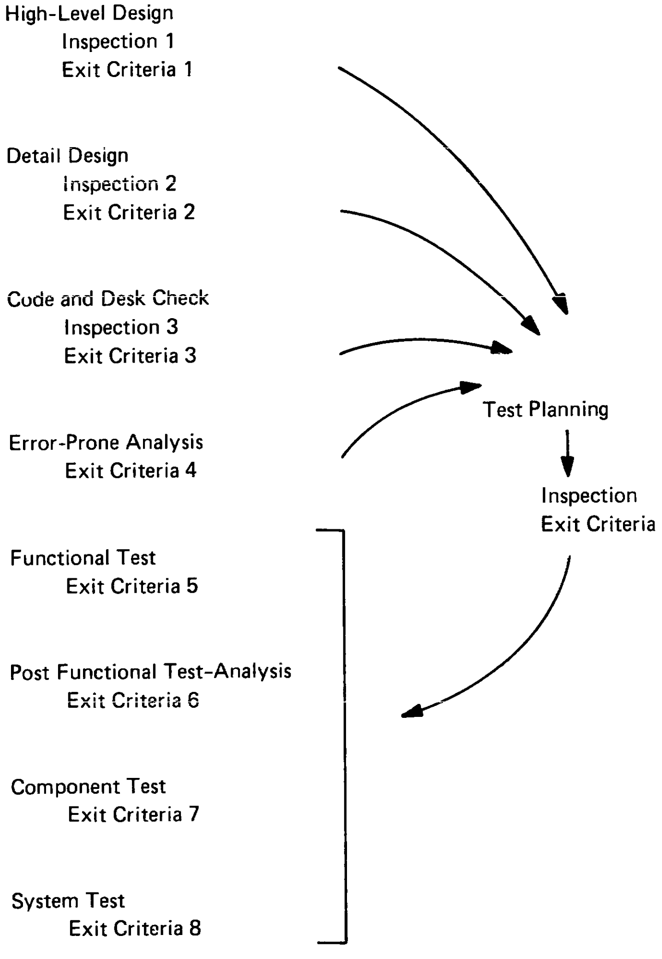


Figure 4—Development cycle of any function

The Error-Prone Analysis step follows the coding phase. The content of the inspection results from High Level Design, Detailed Design, and Code is analyzed. Those areas that required continuous corrections and changes are identified. The reports generated during this step serve two purposes. They will allow the developers to decide whether certain areas should be reworked before entering the Test phase. The decision might require a change to the build plan mentioned earlier. The reports can also be used as additional guideline to the Testing phase in that the error prone areas may require more extensive test effort.

The Post Functional Test Analysis step, as the name suggests, immediately follows the Functional Test. The recorded test results are analyzed. Besides identifying areas which are functionally weak, the data may be used to estimate the number of errors that will occur when the system goes into production. This estimate is measured against a pre-established error tolerance level. Thus the results from Post Functional Test Analysis may be used to determine the quality of the final product.

Due to constraints of available space, we will not discuss each exit criteria. Only the Exit Criteria 1 will be described along with recommended tools for the High-Level Design phase.

The output from the High-Level Design is a design specification in the form of flow-charts and HIPO[9,13] diagrams. The recommended tool here is the structured programming concept in that the flowcharts abide by the rules of structured programming. These will be used for the inspection. The Exit Criteria 1 contains the following:

(1) High-Level Design inspections results logged into an inspection file.
(2) All necessary rework signed-off by managers and logged on the inspection file as complete.
(3) The updated version of the High-Level Design Specification is logged into a project workbook, signed-off by managers, and distributed to (i) project developers, (ii) testers, (iii) documentation and publications groups and (iv) product assurance group (if any).
(4) Each module is assigned an owner.

Note that the required output, High-Level Design Specification, serves, not only as input to the next phase of the development, but also as input for (a) early test planning and (b) documentation.

The Development Cycle (Figure 4) may be viewed as the static component of development environment as every function is produced by going through the same cycle. Its dynamics are shown in the Programming Process Paradigm (Figure 2). Thus the Development Cycle, into which various quality controls are built is the static part of a dynamic Programming Process Paradigm.

SOME JUSTIFICATIONS

In the previous section, we have presented an implementation of quality control in software development. The Inspections, Error-Prone Analysis, and Post Functional Test Analysis are all aimed at attaining the main objective of low error rate. The Exit Criteria which ensures proper control and certain amount of documentation are aimed at achieving the secondary objective of easiness of enhancement and usage.

These objectives of quality control seem to be "user" oriented rather than the developers. We believe that if properly implemented, the developers will benefit just as much. That is schedules will be met, and total development cost will go down. This can be demonstrated by comparing the cost of quality control (in people hours) against the cost of having to correct the errors (in people hours) afterwards.

Figure 5 shows the estimated person hours to inspect code and design. Recall that inspections are conducted with a designer, coder, tester and a representative from publications. One of these may be the moderator. So inspections require four people. Figure 5 shows the cost of quality control in developing different sized products in terms of people-hours based on 10 hr/k for High-Level Design, 15 hr/k for Detailed Design, and 13 hr/k for Code inspections. (k=1,000 lines of executable code)

It is conjectured[10] that on the average a programmer makes 7 errors per 1,000 lines of source code. Assuming that the normal testing eliminates 50 percent of the errors and that with the Inspections and other quality controls the test efficiency goes up to 60 percent, then Figure 6 shows the number of errors left after release. The range of error-elimination efficiency of Inspections is chosen to be 30 percent, 40 percent, 50 percent and 60 percent.

It is estimated that each of these remaining errors costs approximately 70 people hours to fix. This includes identify the problem, make the correction, compile or assemble, test, and check the test. In some cases where redesigning becomes necessary the cost can be much higher. So far, we have assumed unique errors. For each of the unique remaining errors there may be several ill effects, causing one to spend more time in verifying that these are all due to the same unique error. We will, however, use the conservative 70 people hours. The question is whether the cost of correcting these remaining errors to the level that the quality control techniques yield is more than the cost of introducing these techniques into the development. Figure 7 shows the cost of correcting these errors to the levels of different Inspection efficiencies. For example, it costs 105 people hours to bring 3.5 errors down to 2 errors.

	1K	5K	10K	20K	50K
High-Level Design Inspection	40	200	400	800	2000
Detail Design Inspection	60	300	600	1200	3000
Code Inspection	52	260	520	1040	2600
Error-Prone Analysis	5	5	8	10	20
Post Functional Test Analysis	8	10	15	20	30
Total	165	775	1543	3070	7650

Figure 5—Cost in person hr.

	1K	5K	10K	20K	50K
Errors Remaining (Normal)	3.5	17.5	35	70	175
Errors Remaining (with Inspection - 30%)	2.0	10.0	20	40	100
Errors Remaining (with Inspection - 40%	1.7	8.5	17	34	85
Errors Remaining (with Inspection - 50%)	1.4	7.0	14	28	70
Errors Remaining (with Inspection - 60%)	1.1	5.5	11	22	55

Figure 6—Errors in released product

A comparison between Figures 5 and 7 shows that not until we can conduct quality control to the point of 60% error elimination before testing does the introduction of these techniques become economically worthwhile. In reality, a lower level of quality control proficiency would probably suffice, for machine time cost was never introduced into the comparison. Note that the quality control techniques here do not require any machine time. Furthermore, we assumed a fixed 70 people hours for error correction. This resulted in a linear relationship between the size of the product and the error correction cost. A more reasonable relationship would show that the cost of fixing an error would be higher the larger the product as in Figure 8.

For small modules, the error correction percentage of quality control would have to be high before it is economically worthwhile.

	1K	5K	10K	20K	50K
To 30% Inspection Efficiencies	105	525	1050	2100	5250
To 40% Inspection Efficiencies	126	630	1260	2520	6300
To 50% Inspection Efficiencies	147	735	1470	2940	7350
To 60% Inspection Efficiencies	168	840	1680	3360	8400

Figure 7—In people hr.

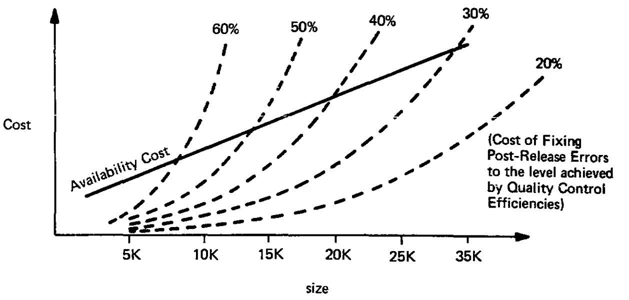


Figure 8

CONCLUSION

In this paper we have introduced a development philosophy called Continuous Integration and a general software devlopment paradigm which can facilitate several Development Cycles. Within any one Development Cycle, a set of quality control techniques may be implemented. With the implementation of quality control, it is shown that the following can be achieved:

(1) lower error rate
(2) ease of maintenance
(3) lower total cost

The economical comparison, although still not precise, does provide a guideline to the management for evaluating quality control techniques.

REFERENCES

1. Aron, J. D., *The Program Development Process*, Addison Wesley, 1974.
2. Baker, F. T., "Structured Programming in a Production Programming Environment," *IEEE Transactions on Software Engineering*, Vol. 1, No. 2, June 1975, pp. 241-252.
3. Basili, V. R. and A. J. Turner, "Iterative Enhancement: A Practical Technique for Software Development," *Proceedings of the 1st National Conference on Software Engineering*, 1975, pp. 56-61.
4. Bratman, H. and T. Court, "The Software Factory," *Computer*, Vol. 8, No. 5, May 1975, pp. 28-37.
5. Dahl, O. J., E. W. Dijkstra and C. A. R. Hoare, *Structured Programming*, Academic Press, 1972.
6. Duke, M. O., "Testing in a Complex Systems Environment," *IBM Systems Journal*, Vol. 14, No. 4, 1975, pp. 353-365.
7. Fagan, M., *Design and Code Inspections and Process Control in the Development of Programs*, TR21.572, IBM Corp., Kingston, N.Y. (Available).
8. Freeman, P., "Toward Improved Review of Software Design," *AFIPS Conference Proceedings*, Vol. 44, 1975, pp. 329-334.
9. HIPO, *Hierarchical Input—Process—Output Documentation Technique: Audio Education Package*, Form SR20-9413, IBM Corp., Gaithersburg, Md. (Available).
10. Kraft, P. and G. M. Weinberg, "The Professionalization of Programming," *Datamation*, October 1975, pp. 169-172.
11. London, R., "A View of Program Verification," *Proceedings of the International Conference on Reliable Software*, 1975, pp. 534-545.
12. Mills, H. D., *Mathematical Foundations for Structured Programming*, Rep. FSC 71-0612, IBM Corp., Gaithersburg, Md. (Available).
13. Nassi, I. and B. Shneiderman, "Flowchart Techniques for Structured Programming," *SIGPLAN* 8, 1973, pp. 12-16.
14. Priven, L., *Managing the Programming Cycle*, TR21.463, IBM Corp., Kingston, N.Y. (Available).
15. Stockenberg, J. E. and A. Van Dam, "STRUCT Programming Analysis System," *Proceedings of the 1st National Conference on Software Engineering*, 1975, pp. 27-36.

A computer performance prediction model

by ROBERT W. OTTO and MARK AUERBACH
Office Chief of Staff, Army
Washington, DC

ABSTRACT

This paper presents a model which departs from the classical queuing theory approach and uses probability to predict computer system throughput. The probability a program desires computer resources (CPU or channels) is obtainable from system measurements or can be estimated. Knowing these probabilities, the multiprograming throughput of a system concurrently executing a number of programs is calculable. The multiprograming model presented is easy to use and requires only the probabilities denoted above. The serial model, reviewed briefly in this paper has been previously published.

INTRODUCTION

The Management Information Systems Directorate (MISD), Office of the Chief of Staff, Army, is responsible for planning hardware and software expenditures for the Army. As the number of Standard Army Management Information Systems grew, Data Processing Installations became saturated. The problem was to validate installation requirements for a hardware upgrade.

The validation process was two-phased. The first phase determined, for the present system, Standard Army Management Information Systems runtime predictors. The second phase extrapolated Standard Army Management Information Systems runtimes to other hardware configurations.

The time needed for extensive computer benchmarking was unavailable. A decision was made to formulate a simple analytic model for determining computer system relative throughput. This paper describes a simple analytic model (in a batch environment) for both serial and multiprograming modes. The latter formulation is probabilistic and both formulations can be exercised with the aid of a desk calculator.

As a follow-up action, when time was available, benchmarks were run. A comparison of theoretical and actual results is presented.

SERIAL MODE FORMULATION

A computer system processing a program resides in one of four mutually exclusive states. These states, a function of the Central Processor Unit (CPU) and Input/Output (I/O), are:

1. $\mathrm{CPU}\cdot\overline{\mathrm{I/O}}$ (CPU busy and no I/O)
2. $\mathrm{CPU}\cdot\mathrm{I/O}$ (CPU busy and I/O)
3. $\overline{\mathrm{CPU}}\cdot\mathrm{I/O}$ (CPU idle and I/O)
4. $\overline{\mathrm{CPU}}\cdot\overline{\mathrm{I/O}}$ (CPU idle and no I/O)

A Resource Utilization Factor (RUF) is the proportion of processing time expended in one of the above states. For a given hardware configuration the RUFs are software dependent, and can be obtained by evaluating performance monitor data.

The serial mode formulation evaluates computer subsystem hardware characteristics. For example, the evaluation characteristics of a disk subsystem are seek time, latency time, and transfer rate. The Resource Correction Factor (RCF, which is more fully explained in Appendix A) is the ratio of a baseline machine characteristic to the same characteristic of the target machine. Associated with each RUF is an RCF.

The Machine Conversion Factor for two computer systems operating in the serial mode (MCFS) is the ratio of time it takes the systems to execute the same program. The MCFS expressed mathematically is:

$$\mathrm{MCFS}=\frac{\mathrm{RUF}_1+\mathrm{RUF}_2+\mathrm{RUF}_3+\mathrm{RUF}_4}{\frac{\mathrm{RUF}_1}{\mathrm{RCF}_1}+\frac{\mathrm{RUF}_2}{\mathrm{RCF}_2}+\frac{\mathrm{RUF}_3}{\mathrm{RCF}_3}+\frac{\mathrm{RUF}_4}{\mathrm{RCF}_4}}$$

where

RUF_k = proportion of time spent in kth state by baseline machine.

$(\mathrm{RUF}_k/\mathrm{RCF}_k)$ = proportion of time spent in kth state by target machine.

By definition the sum of the RUFs for the baseline system is unity and the MCFS equation reduces to:

$$\mathrm{MCFS}=\left(\sum_{K=1}^{4}(\mathrm{RUF}_k/\mathrm{RCF}_k)\right)^{-1} \qquad (1)$$

Example 1: Given the following:

RUF_1 $(CPU\cdot\overline{I/O})=.4$
RUF_2 $(CPU\cdot I/O)=.1$
RUF_3 $(\overline{CPU}\cdot I/O)=.3$
RUF_4 $(\overline{CPU}\cdot\overline{I/O})=.2$

compute the MCFS for a CPU four times faster ($RCF_1=4$) than the original CPU. Utilizing the above equation:

$$MCFS=((.4/4)+.1+.3+.2)^{-1}=1.43$$

This implies CPU replacement results in a target system with throughput 1.43 times greater than the baseline system. RUF_3 and RUF_4 are unaffected because the peripheral subsystem is unchanged. The rationale for equating RCF_2 to unity is as follows:

When the CPU and I/O operations are overlapped the effect on RCF_2 is governed by the smaller RCF involved.

Example 2: With the addition of a peripheral subsystem whose transfer rate is twice as fast, compute the MCFS using the information in Example 1.

$$MCFS=((.4/4)+(.1/2)+(.3/2)+.2)^{-1}=2.0$$

In this example the CPU RCF is four and the I/O RCF is two. The smaller of the RCFs governs and RCF_2 is set equal to the I/O RCF.

MULTIPROGRAMING MODE FORMULATION

In the multiprograming mode, core resident programs are contending for computer resources (Central Processor Unit (CPU) and selector CHannels (CHs)). Contention is the need for a program to acquire a computer resource when the resource is unavailable. The following formulation assumes (1) resource contention is a random event and (2) the probability a program requires a resource and the resource is unavailable, is the intersection of the probability of the program desiring the resource and that at least one other program desires the resource. The Degradation Factor (the DF is more fully explained in appendix B) is the proportionate increase in a program's wallclock time attributed to resource contention. (For purposes of this paper wallclock time is the interval of time a program resides in core.) The DF consists of a CPU contention component and CH contention components.

A computer's operating system services I/O requests on a first come, first served basis. Since I/O requests have no priorities, the DF CH contention components for program i are:

$$DF_{i,j}=CH_{i,j}\left(1-\prod_{\substack{k=1\\k\neq i}}^{N}(1-CH_{k,j})\right) \qquad (2)$$

where

$DF_{i,j}$ = program i contention for channel j.
$CH_{i,j}$ = probability program i requires channel j.
$CH_{k,j}$ = probability program k requires channel j.

Both i and k range from 1 to N (number of programs being processed). Once i is set, k then takes on all other values in the range.

The multiprograming model assumes the scheduling algorithm allocates CPU priority equally to all programs. (This model can be modified for a system having CPU prioritized scheduling algorithm). The DF of the CPU contention component for program i is:

$$DF_{i,c}=CPU_i\left(1-\prod_{\substack{k=1\\k\neq i}}^{N}(1-CPU_k)\right) \qquad (3)$$

where

$DF_{i,c}$ = program i contention for CPU. (c is the number of channels plus unity).

CPU_i = probability program i requires the CPU.
CPU_k = probability program k requires the CPU.

(Recall that both i and k range from 1 to N.)

The DF for program i is unity plus the sum of the contention components, which expressed mathematically, is:

$$DF_i=1+\sum_{c}DF_{i,c} \qquad (4)$$

where, as previously defined:

c = number of contention terms which is number of channels plus unity.

The Machine Conversion Factor for a computer system operating in a Multiprograming mode (MCFMP) is the sum of the inverses of the DFs. (See Appendix B for derivation). Expressed mathematically the MCFMP is:

$$MCFMP=\sum_{i=1}^{k}DF_i^{-1} \qquad (5)$$

where

k = number of programs concurrently being processed.

Example 1: Given the following Resource Utilization Factors (RUFs) for a two channel system:

	Program A	Program B	Program C
RUF_1 $(CPU\cdot\overline{I/O})$	.4	.5	.1
RUF_2 $(CPU\cdot I/O)$	.1	.1	.3
RUF_3 $(\overline{CPU}\cdot I/O)$	.3	.3	.1
RUF_4 $(\overline{CPU}\cdot\overline{I/O})$	.2	.1	.5

and with additional information that the ratios of CH1 and CH2 usage for the above programs are 3:1, 1:1, and 1:3, respectively, compute the MCFMP.

Adding $CPU\cdot\overline{I/O}$ and $CPU\cdot I/O$ yields the proportion of CPU usage. Adding $CPU\cdot I/O$ and $\overline{CPU}\cdot I/O$ yields the proportion of total I/O usage. Performing the indicated additions yields the following:

	Program A	Program B	Program C
CPU	.5	.6	.4
I/O (Total)	.4	.4	.4
I/O (CH1)	.3	.2	.1
I/O (CH2)	.1	.2	.3

CH1 contention components are:

$$DF_{1,1}=.3(1-(1-.2)(1-.1))=.08$$
$$DF_{2,1}=.2(1-(1-.3)(1-.1))=.07$$
$$DF_{3,1}=.1(1-(1-.3)(1-.2))=.04$$

CH2 contention components are:

$$DF_{1,2}=.1(1-(1-.2)(1-.3))=.04$$
$$DF_{2,2}=.2(1-(1-.1)(1-.3))=.07$$
$$DF_{3,2}=.3(1-(1-.1)(1-.2))=.08$$

CPU contention components are:

$$DF_{1,3}=.5(1-(1-.6)(1-.4))=.38$$
$$DF_{2,3}=.6(1-(1-.5)(1-.4))=.42$$
$$DF_{3,3}=.4(1-(1-.5)(1-.6))=.32$$

Degradation Factors for each program are:

$$DF_1=1+.08+.04+.38=1.50$$
$$DF_2=1+.07+.07+.42=1.56$$
$$DF_3=1+.04+.08+.32=1.44$$

and the MCFMP is:

$$MCFMP=\frac{1}{1.50}+\frac{1}{1.56}+\frac{1}{1.44}=2.00$$

Thus, the multiprogramed system executing three programs has twice the throughput of the same configuration executing in a serial mode.

Example 2: Given the following RUFs common to all programs processed:

$$RUF_1(CPU\cdot\overline{I/O})=.4$$
$$RUF_2(CPU\cdot I/O)=.3$$
$$RUF_3(\overline{CPU}\cdot I/O)=.1$$
$$RUF_4(\overline{CPU}\cdot\overline{I/O})=.2$$

compute the MCFS and MCFMP if the original is replaced by a CPU twice as fast ($RCF_1=2$).
The MCFS calculation is:

$$MCFS=((.4/2)+.3+.1+.2)^{-1}=1.25$$

The "new RUFs" ($\overline{RUF}$s) for the faster machine are:

$$\overline{RUF}_k=(RUF_k/RCF_k)(MCFS) \qquad (6)$$

where

$\overline{RUF}_k$ = proportion of time spent in kth state by faster machine.

Thus:

$$\overline{RUF}_1(CPU\cdot\overline{I/O})=(.4/2)(1.25)=.250$$
$$\overline{RUF}_2(CPU\cdot I/O)=.3(1.25)=.375$$
$$\overline{RUF}_3(\overline{CPU}\cdot I/O)=.1(1.25)=.125$$
$$\overline{RUF}_4(\overline{CPU}\cdot\overline{I/O})=.2(1.25)=.250$$

From the $\overline{RUF}$s the probability a program requires the CPU is .625, and the probability a program requires a channel is .50. Given a 3 to 1 ratio for CH1 usage to CH2 usage, (CH1 usage is .375 and CH2 is .125), compute the MCFMP for three programs concurrently processed.
The CH and CPU contention components are:

$$DF_{i,1}=.375(1-(1-.375)^2)=.23$$
$$DF_{i,2}=.125(1-(1-.125)^2)=.03$$
$$DF_{i,3}=.625(1-(1-.625)^2)=.54$$

Since all programs have identical RUFs, the DF for each program is:

$$DF=1+.23+.03+.54=1.80$$

The MCFMP for a system where all programs have identical RUFs is:

$$MCFMP=(MCFS)(k/DF) \qquad (7)$$

where

k = number of programs concurrently being processed.

For this example the MCFMP is:

$$MCFMP=1.25(3/1.80)$$
$$MCFMP=2.08$$

This example demonstrates that, given a set of representative system RUFs and approximating the average number of programs concurrently processed, the MCFMP of a system can be determined.

FORMULATION REFINEMENT

If the initial benchmarks are executed in both a serial and multiprogramed mode, equation 4 is written as:

$$\overline{DF}_i=1+(T)\left(\sum_c DF_{i,c}\right) \qquad (4')$$

where

T = a proportionality constant

Replacing DF with $\overline{DF}$ equation 5 becomes:

$$MCFMP=\sum_{i=1}^{k}\overline{DF}^{-1} \qquad (5')$$

Having an empirical value for MCFMP equation 5′ is initially solved for T. The MCFMP for the target systems are computed as demonstrated in the multiprograming mode formulation section with equations 4 and 5 replaced by equations 4′ and 5′.

LIMITATIONS

The multiprograming formulation is valid for concurrent processing of at least two and possibly three programs (expected queue lengths are very short). When concurrently processing four or more programs significant queues will form, and a more complex queuing theory formulation is necessary.

RESULTS

From hardware monitoring, the average Resource Utilization Factors (RUFs) and channel usage for Standard Management Information Systems executing on an IBM 360/30 were obtained. These RUFs are:

$\mathrm{CPU}\cdot\overline{\mathrm{I/O}}=.55$
$\mathrm{CPU}\cdot\mathrm{I/O}=.06$
$\overline{\mathrm{CPU}}\cdot\mathrm{I/O}=.18$
$\overline{\mathrm{CPU}}\cdot\overline{\mathrm{I/O}}=.21$

Channel 1 accounts for 78 percent of I/O activity and channel 2 accounts for 22 percent.

Using an instruction mix it was determined that the average instruction time for an IBM 360/30 is 23.6 microseconds; for an IBM 360/40 13.6 microseconds; and for an IBM 360/50 6.5 microseconds. The following chart compares the calculated Machine Conversion Factor (MCF) using the model, to the actual benchmark MCF.

System	Calculated MCF (T=1)	Calculated MCF (T=1.3)	Benchmark MCF
30	1.0	1.0	1.0
30MP (2 programs)	1.4	1.3	1.3
40	1.3	1.3	1.3
40MP (2 programs)	2.0	1.9	1.7
50	1.7	1.7	1.6
50MP (2 programs)	2.8	2.6	2.6

The calculated MCFs were used to validate hardware requirements at 36 Army Data Processing Installations in order to initiate a procurement for the necessary upgrades.

APPENDIX A

RESOURCE CORRECTION FACTORS

A Resource Correction Factor (RCF) is the ratio of a baseline measurement characteristic to the measurement of the same characteristic of the target computer system. For example, calculation of a computer system's throughput for replacement of a disk subsystem requires two RCFs. One RCF is the ratio of transfer rates and is used when I/O is occurring; the other RCF is the ratio of seek time plus latency time and is used when the device is busy only and no I/O is occurring.

Calculation of a computer system's throughput for replacement of a Central Processor Unit (CPU) requires an analysis of software. Knowing the instruction repertoire of a computer, the incidence of use of the various instructions can be obtained by software monitoring. To obtain a CPU RCF:

(1) Multiply each instruction's frequency of use by the instruction's execution time.
(2) Sum the results of step one.
(3) Take the ratio of step two for the baseline CPU to the target CPU.

Example: Given the following:

		BASELINE CPU		TARGET CPU	
INSTRUCTION	FREQ	TIME	(FREQ) (TIME)	TIME	(FREQ)(TIME)
Load	8	4	32	1	8
Store	4	3	12	2	8
Branch	6	7	42	5	30
Add	2	2	4	2	4
			90		50

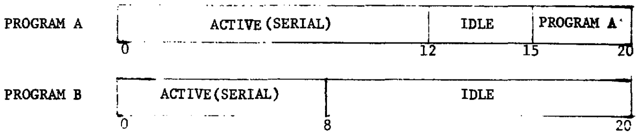
The CPU RCF is 90/50 = 1.8

APPENDIX B

DEGRADATION FACTOR

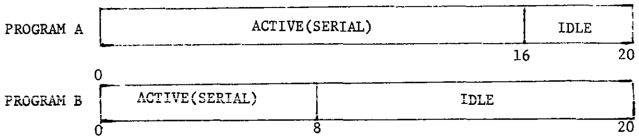
Multiprograming is the concurrent processing of two or more programs and the sharing of computer resources by these programs. Multiprograming increases wallclock time (time a single program resides in core) and increases total computer system throughput. The Degradation Factor (DF) is the ratio of wallclock time for a program processing on a computer operating in a multiprograming mode to a serial mode. The Machine Conversion Factor for the Multiprograming mode (MCFMP) is the increase in system throughput for a system operating in a multiprograming mode opposed to a serial mode. The MCFMP can be calculated by taking the sum of the inverses of the DFs.

Example 1: Given that programs A and B executed with serial wallclock times of 12 units and 8 units respectively, and multiprogramed wallclock times of 15 units and 20 units respectively, compute the DFs and the MCFMP.



The DF for program A is 15/12=1.25 and the DF for program B is 20/8=2.5. The MCFMP is (1/1.25)+(1/2.5)=1.2. The multiprogramed system has 1.2 more throughput than the serial system.

Example 2: Assuming both programs in the previous example required exactly 20 units of wallclock time when multiprogramed and each program retained its ratio of active to idle time, compute the MCFMP.



The total active (serial) processing time of programs A and B, divided by the wallclock time is the MCFMP. The MCFMP is (16+8)/20=1.2, thus verifying the method used in Example 1.

A queueing network model for the effect of data compression on system efficiency*

by ALAN JAY SMITH
University of California at Berkeley
Berkeley, California

ABSTRACT

Data compression is often employed in data base systems to reduce the volume and therefore the cost of large online data bases. The use of data compression is normally assumed to imply a tradeoff—storage space is decreased in return for which the CPU is required to spend time coding and decoding the data, with a consequent decline in system throughput. We employ queueing network models with classes of customers to show that the effect of data compression on system efficiency can be calculated. Our calculations indicate that under some circumstances, data compression may actually increase system throughput by decreasing input/output transfer time and/or increasing the useful processing per data record. We formulate and solve several models to indicate the power of our technique.

INTRODUCTION

Large data base systems exist or are being developed for applications including library information retrieval, such as the Spires system at Stanford University and the Intrex System[1] at M.I.T., for business data management, such as IBM's IMS system[2] and for military applications such as the Navy Fleet Material Support Office.[3] In all data base systems, the cost of the online storage devices, be they disks, drums or more exotic devices such as the photostore, can be expected to be a very large fraction of the hardware cost of the system. In standard computer systems, examination of manufacturers price sheets suggests that the memory hierarchy accounts for about one half of the cost of the hardware; in a data base system the storage devices can be expected to account for even a larger fraction of the system cost.

Because of the high cost of maintaining information online in a computer system, the application of data compression has become increasingly attractive. Normally, the use of data base compression implies a tradeoff; the decrease in storage space and cost is achieved only at a substantial expense in coding and decoding the data base. In this paper we develop a model for the effect of data compression on system efficiency. We show that under certain circumstances, system throughput may actually be increased through the use of data compression.

In the next section of this paper we briefly review some of the more common and effective methods of compression that are applicable to computer files. The third section summarizes the mathematical solution to a very general class of queueing networks. We analyze a simple model of data compression in the fourth section and present numerical results. Other, more sophisticated models that take into account more of the characteristics of real systems are presented in the fifth section and the mathematical formulation for some of these models is given.

REVIEW OF DATA COMPRESSION METHODS

Information theory

In his classic paper in 1948, Claude Shannon created the subject of information theory.[4] He defined entropy as a measure of information and showed how to estimate mathematically the amount of information contained in a message or set of messages. Although his formulation is completely general, the direct application of his results is difficult; most data compression schemes employ simple but effective redundancy removal techniques that are not directly derived from (although they are certainly related to) information theory considerations. Some of these techniques are briefly discussed below.

Null suppression

Many files are largely composed of "nulls," zeros in numeric fields and blanks in character fields. These nulls are inserted either because of empty fields or because the field length is larger than the data item

* Research sponsored by the Joint Services Electronics Program Contract F44260-71-C-0087.

it contains. Simply omitting these nulls (and inserting field separators where necessary) can eliminate a great deal of redundancy and can substantially compress the file.

Run length coding

Run length coding involves coding runs of identical data items (or groups of data items) as a count of the number of occurrences followed by a sample of the item(s). Thus a run of five zeros might be encoded as "50." This technique is most effectively employed to run length encode nulls (zeros or blanks) but may also be useful when other characters are repeated.

Statistical encoding

Statistical encoding techniques are directly derived from information theoretic considerations and take advantage of the varying probabilities of occurrence of characters or groups of characters. Shorter codes are used to represent the most common items and longer codes the less frequent. Huffman devised a simple and optimum method of statistical encoding. Huffman coding is almost always used only on a character by character basis for ease of implementation and thus generally produces only limited data compression.

Pattern substitution

One of the most effective methods of compression is pattern substitution. Character or digit strings of length greater than one are replaced with shorter strings; these shorter strings can be assigned using Huffman's algorithm. Thus coding and decoding dictionaries are created and are used to translate to and from the compressed form.

Front and back compression—differencing

When a field of a set of records is stored in sorted form, only the differences between records need be retained. Thus in the phone book, the whole page of "Smith's" could be shortened by only giving first and middle names. If the loss of information is no problem, as in index structures, only that part of a record that serves to distinguish it from its neighbors need be retained. Thus the trailing part(s) of a record field can often also be deleted.

QUEUEING NETWORKS WITH CLASSES OF CUSTOMERS

In this section we summarize the mathematical solution to a very general class of queueing networks. The material is entirely drawn from Baskett et al.[5] and the reader is referred to that paper for a more complete presentation.

A queueing network is a model of the following type: There are a number of service stations (e.g., computers, toll booths, checkout counters, work stations), each with one or more individual servers. Behind each service station (we use the term server interchangeably with service station except where confusion is likely to result) a queue forms. Customers arrive in the queue behind the service station, are served according to one of the queueing disciplines discussed below, and then each customer, with a certain probability, either joins the queue behind another server or leaves the system. Customers can also arrive at a given rate at certain servers from outside the system.

The generality of the queueing network model can be extended by allowing customers to be of different types (from different "classes") and by allowing each type of customer to have different routing probabilities and different service requirements. Our models will depend heavily on the use of classes of customers.

Servers (service stations) may be of one of four types for the product form solution (of equation (2)) to the queueing network to be valid; we present only those three types of servers that are relevant to our models.

Types of servers

Type 1: There is a single server at a service center, the customers are served in first come, first served order, all customers have the same service time distribution at the service center and this distribution is the negative exponential. The service rate may depend on the number of customers queued and waiting for the server.

Type 2: There is a single server at the service center and the processor is shared among all customers waiting for the processor (i.e., each customer receives service at a rate of 1/n when there are n customers). The service time distribution can vary by customer class and may be arbitrary. Type 2 servers are frequently used to model the CPU in a computer system and represent quite accurately the operation of a processor that is scheduled in a round robin manner with a short quantum.

Type 3: The number of servers in the service center is greater than or equal to the maximum number of customers that can ever be queued at this server. Each class of customer may have a distinct service time distribution of arbitrary form. Users at their teletype terminals are well described as Type 3 servers.

All service distributions for Type 2 and Type 3 servers must have rational Laplace transforms, but this requirement is no barrier since any non-pathological distribution can be approximated arbitrarily closely by one with a rational transform.

Let us define $1/\mu_i(j)$ to be the mean service time at the i^{th} server, which is of Type 1, when there are j customers queued or in service. Let $1/\mu_{ir}$ be the mean service time of a customer of class r at server i for a server of Type 2 or 3. μ is then the "rate" of service, and is equal to the reciprocal of the mean.

$p_{i,r;j,s}$ is defined as the probability that a customer of class r, when it completes service at server i will change to class s and go next to server j. Let n_{ir} be the number of customers of class r at service center i. Let $n_i = \sum_{r=1}^{R} n_{ir}$. Let N be the number of customers, R the number of classes and M the number of service stations.

Our models will have exactly one chain; that is, a customer at server i in class r can in some manner reach server j and be in class s if and only if it is possible for a class s customer to ever be at server j. Thus all possible customer states (i,r) communicate with all others. In this case we can define a set of equations which specify the variables e_{ir} as follows:

$$\sum_{\forall i,r} e_{ir}p_{i,r;j,s} + q_{js} = e_{js} \quad \text{for all j, s} \tag{1}$$

where q_{js} is the rate of customer arrivals of class s to server j from outside the system (exogenous arrivals). If $q_{js}=0$ for all j, s, then the e_{ir} are determined to within a multiplicative constant; otherwise they are specified uniquely. The e_{ir} can be interpreted as the relative (or absolute) arrival rate of class r customers to service center i.

The steady state solution for the queueing network we have described has a very convenient form; the overall steady state solution is the product of terms for each server in the network. Let y_i be the state of the i^{th} server (number of customers, etc., see below). Then

$$P(S=y_1,y_2,\ldots,y_N) = Cd(S)g_1(y_1)g_2(y_2)\ldots g_N(y_N) \tag{2}$$

where C is a normalizing constant, d(S) is a function of the number of customers in the system and g_i is a function depending on the type of server i. If service center i is of Type 1 then

$$g_i(y_i) = n_i!\left[\prod_{r=1}^{R}\frac{1}{n_{ir}!}[e_{ir}]^{n_{ir}}\right]\prod_{j=1}^{n_i}\frac{1}{\mu_i(j)} \cdot \tag{3}$$

If service center i is of Type 2, then

$$g_i(y_i) = n_i!\prod_{r=1}^{R}\frac{1}{n_{ir}!}\left[\frac{e_{ir}}{\mu_{ir}}\right]^{n_{ir}} . \tag{4}$$

If service center i is of Type 3, then

$$g_i(y_i) = \prod_{r=1}^{R}\frac{1}{n_{ir}!}\left[\frac{e_{ir}}{\mu_{ir}}\right]^{n_{ir}} . \tag{5}$$

We note that only the mean service times $(1/\mu)$ appear in the above expressions; thus the steady state distributions are independent of the form of the service time distributions for customers at servers of Types 2 or 3.

We will, for simplicity, consider only models in which the number of customers (N) is fixed. This represents a situation in which the degree of multiprogramming is constant. The extension to open systems (ones with exogenous arrivals and departures) is straightforward, but it adds only additional parameters to our models and provides no new information. For such a closed system, d(S) = 1.

The normalizing constant may be calculated as follows:

$$C = 1/\sum_{\text{all feasible S}}[g_1(y_1)g_2(y_2)\ldots g_N(y_N)]. \tag{6}$$

That is, the sum of all steady state (equilibrium) probabilities must be 1. The sum is taken over all "possible" states S. Possible states are those in which the total number of customers in the system is equal to the proper figure (in this case N) and all class/server and class/number in class combinations are correct. Thus if it is impossible for a class 3 customer to be at server 5, then any state with a class 3 customer at server 5 is not feasible, and the summation must not include such a term.

We note that for open systems, it is possible to get a closed form expression for C, the normalizing constant.

Prior to the development of solutions for queueing networks with classes of customers, queueing networks were limited to one class of customer. It was thus impossible to differentiate among customers when they arrived at a server. By allowing each customer to be of a different class, it is possible to route each customer differently in the network. By allowing customers to change classes, it is possible to route the same customer through the same service center more than once and to have the customer display different behavior, both in service requirements and in departure branching probabilities. In the paper by Baskett et al.,[5] it can be seen that this ability to differentiate among customers yields measurably different steady state probabilities in comparison with the approximations employed previously.

A SIMPLE MODEL FOR DATA BASE SEARCH

With a single, FCFS, storage device

In this section we will describe, solve and present numerical results for what is a simple model for data base operation with compression. We have chosen this particular model for two reasons: It illustrates virtually all of the interesting features of our technique and our ideas and it contains a small number of parameters that can be varied intelligently. Formulation of more sophisticated or complex models is straightforward and two such are presented in the next section.

We will consider a data base system in which the only operation to be modeled is the search operation. The system will be multiprogrammed to a constant factor of N. Each of the N processes (customers) will follow the following sequence of steps repetitively:

1. Computation—The CPU will perform the necessary computations on the record just retrieved, and then will perform the operations necessary to initiate the next read operation (process in class 2).
2. The process will be suspended while a record is read from the secondary storage device and transferred to a buffer area in main memory (process in class 1).
3. The CPU will perform the decompression operations necessary to translate the record into decompressed form (process in class 1). Operation continues again at Step 1.

This sequence of operations is diagrammed in Figure 1 where a number of useful parameters of the model are also shown. We define these parameters below as well.

$1/\mu_1$ The mean time to transfer a record from secondary storage to main storage.

$1/\mu_{21}$ The mean time to decompress a stored record.

$1/\mu_{22}$ The mean time to perform the computations in Step 1 above.

Server 1, the secondary storage device, will be modeled as a Type 1 server. That is, customers will be serviced in first come, first served order and the service time distribution will be the negative exponential. This formulation should be an adequate approximation for any I/O device that services the customers in the order of arrival with no parallel operation (e.g., disk, drum, extended core storage).

Server 2, the CPU, will be modeled as a Type 2 server; the customers will be serviced in processor sharing model. We note that since the model yields the same steady state probabilities independent of the service time distribution, the highly skewed service time distributions often found in computer systems cause us no inconvenience. Such skewed distributions are, in fact, unlikely for the applications that we discuss.

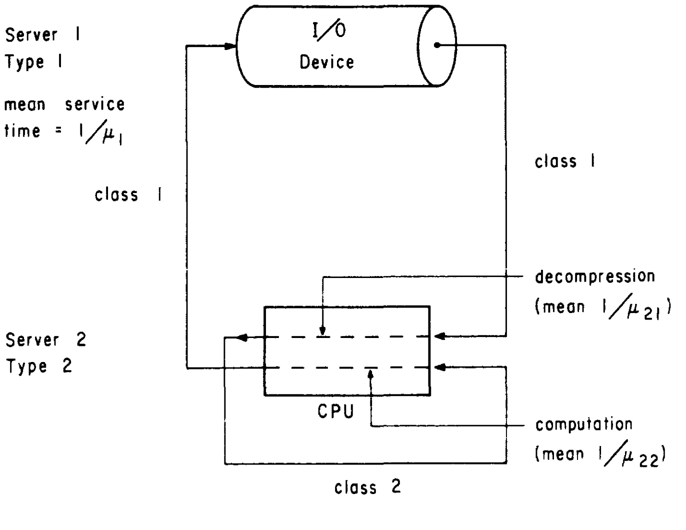


Figure 1—Model 1

From the above descriptions, it is evident that for the transition probabilities we have:

$$\begin{aligned} &p_{1,1;\,2,1}=1\\ &p_{2,1;\,2,2}=1\\ &p_{2,2;\,1,1}=1\\ &p_{i,r;\,j,s}=0 \text{ for all other i, j, r, s.}\end{aligned} \tag{7}$$

From equation (1) for the e's, it is clear that the e's are all equal and may vary (jointly) by a multiplicative constant. For convenience we choose that constant so that

$$e_{i,r}=1 \text{ for all i, r.} \tag{8}$$

For only two servers, the product form solution is of the form

$$P(S)=Cg_1(y_1)g_2(y_2) \tag{9}$$

where

$$\begin{aligned} g_1(y_1)&=n_1!\left[\prod_{r=1}^{2}\frac{1}{n_{1r}!}\right]\left[\prod_{j=1}^{n_1}\frac{1}{\mu_1}\right]\\ &=\frac{1}{\mu_1^{n_1}}\end{aligned} \tag{10}$$

and

$$g_2(y_2)=\frac{n_2!}{n_{21}!n_{22}!\mu_{21}^{n_{21}}\mu_{22}^{n_{22}}}. \tag{11}$$

Thus

$$P(S)=C\frac{n_2!}{\mu_1^{n_1}n_{21}!n_{22}!\mu_{21}^{n_{21}}\mu_{22}^{n_{22}}}. \tag{12}$$

The computation for the normalizing constant is indicated in equation (6).

The parameter of interest in this model will be the fraction of time that the CPU spends processing class 2 customers, since that is the computation that is independent of the amount of compression that is employed. If we assume that this computation (on class 2 customers) represents the "useful" work of the system, then we are actually measuring the throughput of the system. The figure of merit is thus

$$\text{throughput}=\sum_{\substack{\forall\, S \text{ such that}\\ n_2>0 \text{ and } S\\ \text{is feasible}}} P(S)\cdot\frac{n_{22}}{n_2}. \tag{13}$$

Although it is simple to write down a closed form expression for the throughput of the system, the large number of terms prevents any clear or easy interpretation of the resulting formula. Symbolic optimization techniques, which are applicable to the models we describe in this section, are likewise infeasible. Attempts to determine the optimum degree of compression, where the effects of compression are indicated in equation (13), were unsuccessful using the Macsyma[6] symbol manipulation facility. For this reason we present only numerical results, as indicated in Figures 2, 3, 4, 5, 6, 7, 8 and 9.

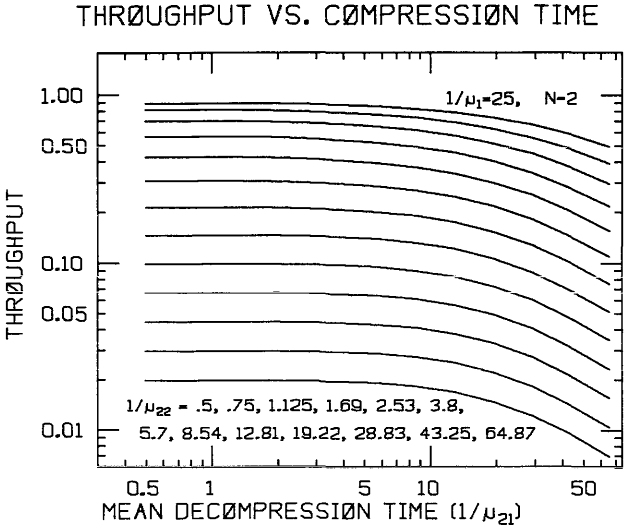


Figure 2

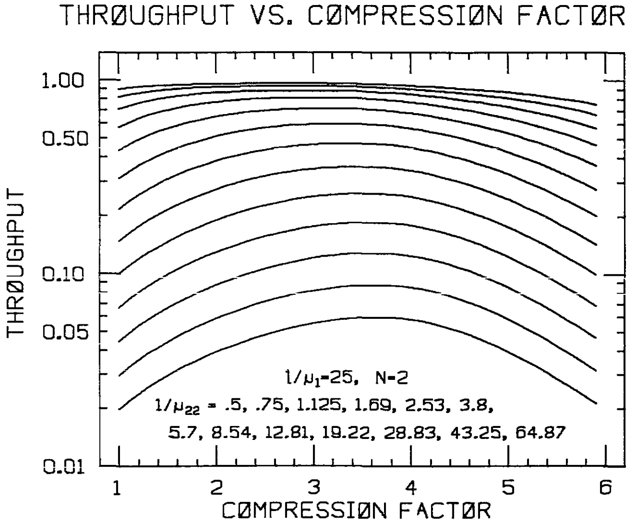


Figure 4

In Figures 2 through 5, the mean service time at server 1 (the secondary storage device) has been taken to be 25 milliseconds. Figures 2 and 4 display computations for a system with a degree of multiprogramming (N) of two; the same data is presented in Figures 3 and 5 for a multiprogramming level of five.

Both Figures 2 and 3 give the system throughput, measured as the useful processor utilization (equation (13)), for a system in which we have varied the mean decompression time ($1/\mu_{21}$) for different values of the mean useful processing time ($1/\mu_{22}$). The values of $1/\mu_{22}$ were varied from .5 to 64.87 milliseconds; each value differs from neighboring values by a factor of 1.5. We note that in the computations presented in Figures 2 and 3, we have allowed ourselves no benefit from the use of compression; compression has neither increased the amount of useful processing per record nor decreased the amount of time to transfer a record. Despite this, it is only when the decompression time exceeds about one-third of the input/output service time (transfer+latency) that the system throughput declines significantly. This decline, and the point at which it begins, is surprisingly independent of the amount of normal processing required per record ($1/\mu_{22}$). This is accounted for by three observations: for small useful processing, the system has substantial excess capacity available for compression and decompression, for large useful processing, the compression

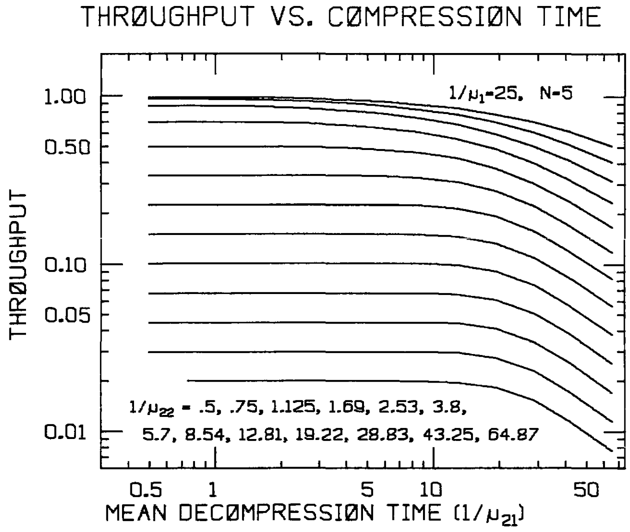


Figure 3

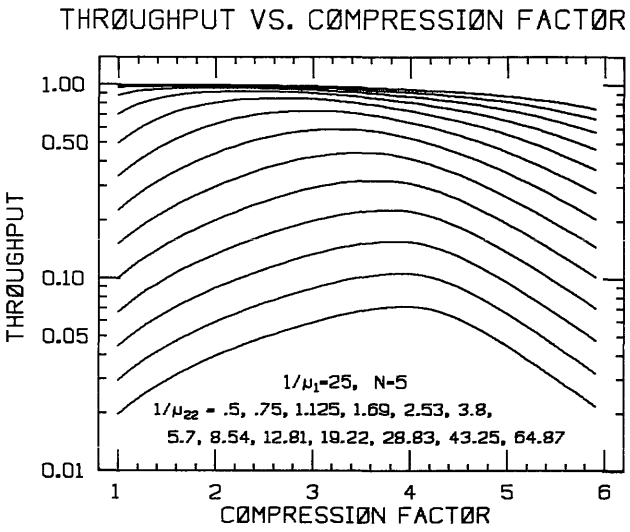


Figure 5

time is a small added increment and is only marginally apparent. In both cases, the effect of the decompression is masked by the input/output service time.

In computing the data displayed in Figures 4 and 5, we have taken a rather more optimistic attitude. We assume here that with the addition of data compression, the size of the record transferred and the consequent input/output service time is unchanged. The information content of the record is therefore increased by a factor equal to the degree of compression D. The usefulness of this latter effect applies to those cases in which we are performing a search; simple lookup operations to a known location within a record result in no increase in useful processing time when the record content is increased. We will let $1/\mu_{22}{}^*=D/\mu_{22}$, where $1/\mu_{22}{}^*$ is the effective mean useful processing time; on the figures we give only the values of $1/\mu_{22}$, which are unadjusted for the compression factor.

The value for the mean decompression time, $1/\mu_{21}$, was computed as follows:

$$1/\mu_{21}=e^{D-1}-.9. \qquad (14)$$

This expression (equation (14)) represents our intuitive idea of the cost of data compression; we have no set of measurements to justify it. We can observe its reasonableness when we consider the experiments by Shannon.[7] Shannon attempted to measure the information content of English. When considered on a letter by letter basis, English requires about 4.14 bits/letter for optimum encoding; for equifrequency coding, it requires 4.7 bits/letter. When coded in groups of two letters (digrams), English needs 3.56 bits/letter. Coding English word by word required about 2 bits/letter. Shannon then conducted experiments with human subjects in which he asked them to predict the next letter of a text, given the preceding letters. This experiment yields a figure of about 1 bit/letter. We observe that each additional reduction of text size required substantially more processing time and larger tables; the final and most effective form of compression involved the long range syntactic structure of English and its semantic content as well. Thus an exponential increase in compression/decompression time with an increase in the degree of compression is reasonable.

It can be seen from both Figures 4 and 5 that the use of compression has resulted in an increase in the system throughput of better than a factor of two for the best value of the degree of compression. The relation of the effect of compression to throughput is most evident in Figure 5 where the peak in each curve moves to the left as the value of $1/\mu_{22}$ increases; compression becomes increasingly less useful as the system utilization climbs until for almost full utilization compression is of no use at all; it simply appears as overhead.

We observe in comparing Figures 2 to 3 and 4 to 5 that the change in the degree of multiprogramming has almost no effect; the shape of the curves is only slightly changed as are the numerical values displayed. This effect can be accounted for by remembering that the bottleneck in the system is the input/output device, which is a first come, first served, server.

A many spindle parallel service system

In our model, which we analyzed earlier, we postulated a single secondary storage device which processed requests for information in a first come, first served manner. Such systems exist; an example is the Ingres System[8] which currently uses a single spindle disk drive to hold all data files. At the other end of the spectrum, one can imagine a system with a very large number of spindles, overlapped seeks, drums with sector queueing and multiple channel controllers to minimize channel interference.[9] We can approximate the operation of such a system with a very simple modification to our model of Figure 1. We let server 1, the secondary storage device, be of Type 3 rather than Type 1. A Type 3 server is one in which the number of individual servers at the service station equals or exceeds the number of customers in the queueing network. For a large number of spindles and a small degree of multiprogramming, this provides an excellent approximation which allows great computational efficiency.

We replace the function $g_1(y_1)$ as defined in equation (10) with the function $g_1(y_1)$ as defined in equation (15), below:

$$g_1(y_1)=\frac{1}{n_1!\mu_1^{n_1}}, \qquad (15)$$

$g_2(y_2)$ is as defined in equation (11); thus the expression for the state probability is

$$P(S)=C\frac{n_2!}{n_1!n_{21}!n_{22}!\mu_1^{n_1}\mu_{21}^{n_{21}}\mu_{22}^{n_{22}}} \qquad (16)$$

where the normalizing constant C is defined as in equation (6).

Figures 6, 7, 8 and 9 correspond to Figures 2, 3, 4 and 5, but employ the model formulation of this section. As before, the mean secondary storage fetch time is 25, the mean useful processing time is varied from .5 to 64.87 and the degree of multiprogramming (N) takes on the values 2 and 5.

Figures 6 and 7 provide an interesting contrast to Figures 2 and 3 in two respects. We see that the throughput drops off almost immediately with the introduction of compression; a dropoff that wasn't apparent until the mean decompression time reached approximately 8 in Figures 2 and 3 is apparent for values as small as 2 in Figures 6 and 7. This effect is expected; the secondary storage device was the bottleneck in the system in our earlier model, but it is much less so here. Compression, therefore, has a far more deleterious effect.

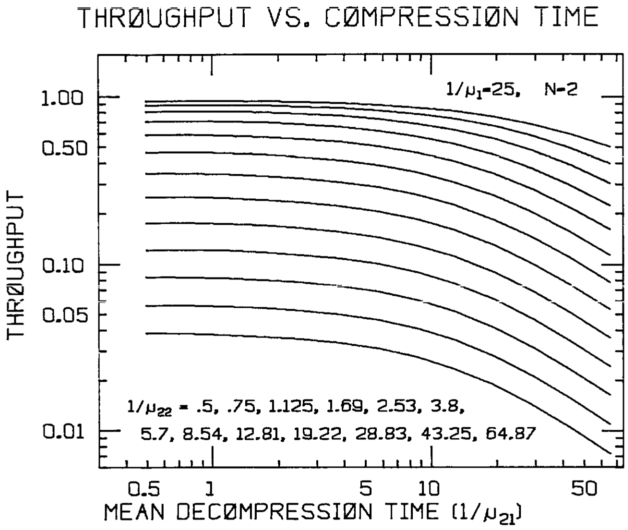


Figure 6

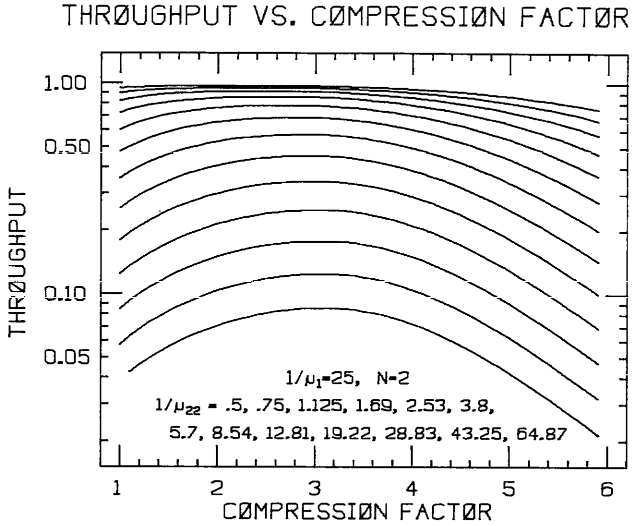


Figure 8

The ability of the storage device to process requests in parallel also introduces a difference between Figures 6 and 7 that wasn't apparent when comparing Figures 2 and 3. The increase in the degree of multiprogramming has resulted in a substantial increase in the system throughput.

The differences between Figures 2 and 3 and Figures 6 and 7 are repeated when one compares Figures 4 and 5 with Figures 8 and 9. The optimum degree of compression has decreased, since the system very much reflects the effect of the processing time required for decompression. This is most pronounced in Figure 9 in which the system can be seen to attain high throughput through multiprogramming to a degree of five. A large increase in throughput is also observed when the degree of multiprogramming increases in going from Figure 8 to Figure 9.

Conclusions

The model described in this section has given us insight into the effect of data compression on system operation. It can be seen that when the processing time required for data compression is sufficiently small, system performance will only deteriorate slightly. When one takes into account the possible increase in processing time per record fetched when compression is introduced, system throughput may increase very substantially. By substituting into our model the

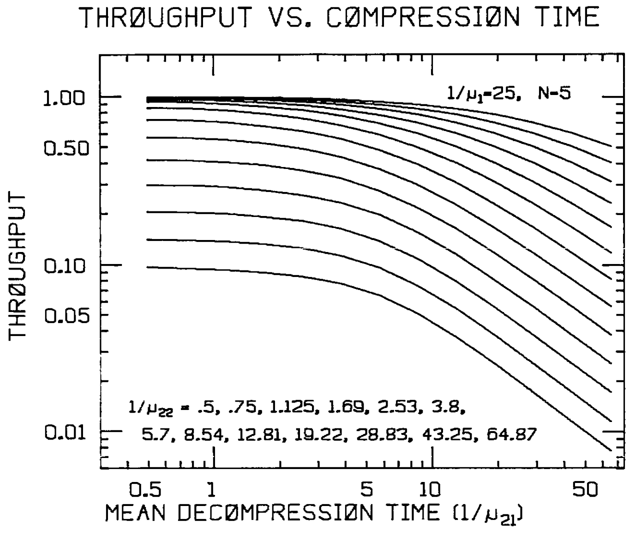


Figure 7

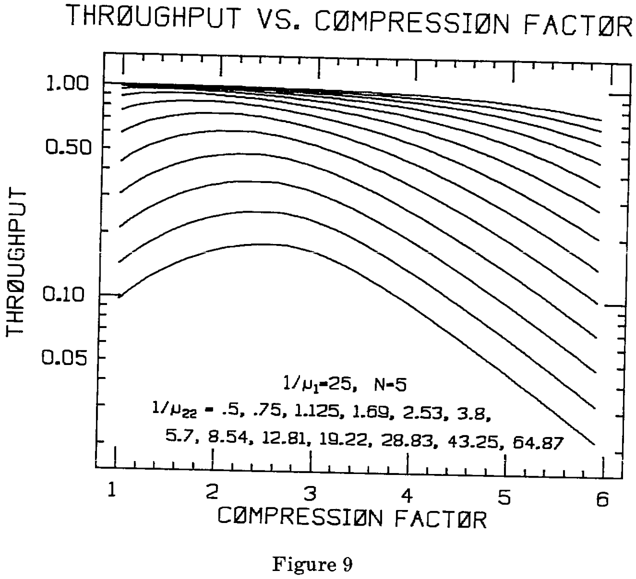


Figure 9

measured values for μ_1, μ_{21}, μ_{22} and N, one can obtain a good estimate of the exact effect of data compression.

In addition to giving us useful results, our analysis has provided a very simple example of a general and powerful technique; that of system modeling with queueing networks with classes of customers. In the next section we indicate some of the power of our techniques with further models.

OTHER MODELS

In the last section we presented a model and its solution for the operation of a data base system using data compression. In this section we describe two more models, of increasing complexity, which capture features often found in data base systems. Our purpose is to indicate the generality that the queueing network formulation for the efficiency problem permits and to demonstrate the ease with which a system of substantial complexity can be modeled.

Both of the models of this section will be primarily concerned with update operations. The first will assume, as did the model of the previous section, that the file records can be located in one operation; the second model of this section will consider the use of a tree structured index.

Model 2

Every process will repetitively follow the following sequence of steps:

1. Perform whatever computation is necessary to choose a record to read (process in class 1).
2. Read a record from the I/O device (class 1).
3. Decompress the record just read (class 2).
4. Perform any desired computations and update to this record (class 3).
5. Recompress the record after computation and updating is complete (class 4).
6. Write the record back into the I/O device (class 2).

A possible extension to this model which may be straightforwardly modeled is one in which, with a certain probability p, the update of the record caused it to expand in size by an amount sufficient to require that an overflow record be written.

In Figure 10 we diagram Model 2 as we have described it.

The solution to this model is simple. The e's (of equation (1)) are again equal to 1, since all of the $p_{i,r;j,s}$'s are equal to one.
Then

$$g_1(y_1) = \frac{n_1!}{n_{11}!n_{12}!\mu_1^{n_1}} \tag{17}$$

and

$$g_2(y_2) = \frac{n_2!}{n_{21}!n_{22}!n_{23}!n_{24}!\mu_{21}^{n_{21}}\mu_{22}^{n_{22}}\mu_{23}^{n_{23}}\mu_{24}^{n_{24}}} \tag{18}$$

The same computations as were used in the previous section suffice to determine the normalization constant C. The figure of merit becomes the fraction of time the CPU spends servicing customers in classes 1 and 3. Therefore, we determine

$$\sum_{\substack{\forall S \text{ feasible} \\ n_2>0}} \frac{n_{21}+n_{23}}{n_2} P(S) \tag{19}$$

which may be calculated quickly on a computer.

It is possible to take into account additional complexities in the operation of the system and we indicate one such possible model in Model 3. This model represents a system with three secondary storage devices, each of which may have a different service rate. The records are stored using a tree structured index, as is used by IBM in its ISAM access method.[2] The index in the case of this model is two levels deep; thus three read operations are needed to find a record, given the primary key. We assume that records, once updated, are rewritten to their original location without the need to access the index structure. The probabilities of finding an index or data record on a given device are specifiable at will in the model. We assume that the placement of index records is independent of the placement of data records; by the addition of a large number of additional classes, this restriction can be removed. The degree of multiprogramming can be specified as preferred or the system can be made into an open one as indicated earlier.

The behavior of a process is as follows:

1. Perform the necessary computations in order to choose a record to read (class 1).

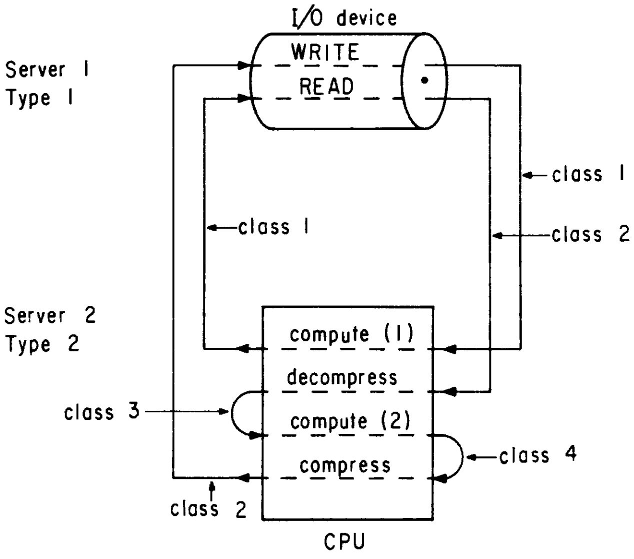


Figure 10—Model 2

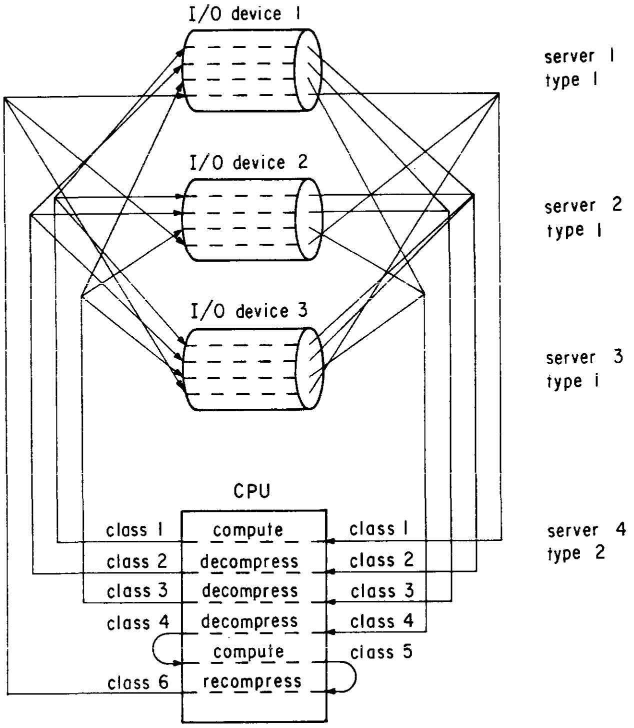


Figure 11—Model 3

2. Read an index record, level 1 (class 1).
3. Decompress the index record (class 2).
4. Read an index record, level 2 (class 2).
5. Decompress the index record (class 3).
6. Read the data record (class 3).
7. Decompress the data record (class 4).
8. Perform whatever update operations and computations on the recently retrieved record that are desired (class 5).
9. Recompress the data record (class 6).
10. Write the data record back on the secondary storage device (class 4).

More complex storage structures and system configurations can easily be dealt with by increasing the number of devices and when necessary, the number of customer classes in order to differentiate between customers and between different operations on the same customer at the same server.

We should mention briefly that as the number of servers and the number of classes of customers increases, the number of system states grows very quickly and it may be computationally infeasible to analyze some systems. Some efficient methods for computing these state probabilities in queueing networks with classes of customers are described by Muntz and Wong.[10]

CONCLUSION

In this paper we have demonstrated the use of queueing networks with classes of customers for modeling the effect of data compression on system efficiency. The simple models developed in section IV indicate that even when compression results in no increase in useful processing time per record or reduction in record retrieval time, it may not significantly impair system performance. When the increase in useful processing per record is considered, compression can be seen to substantially increase system throughput. Earlier, we formulated two more sophisticated models that indicate the generality possible with our modeling technique. We expect that the methods demonstrated in this paper will be of great use to system designers considering the use of data compression for computer file systems.

ACKNOWLEDGMENTS

Partial support for this research was provided by the Joint Services Electronics Program under contract F44260-71-C-0087. Computer time was furnished by the Energy Resources Development Administration (née AEC) under grant AT-(04-3)-515.

REFERENCES

1. Overhage, C. F. J. and J. F. Reintjes, "Project Intrex: A General Review," *Information Storage and Retrieval 10,* 5/6, pp. 157-188.
2. Date, C. J., *An Introduction to Data Base Systems,* Addison-Wesley, Reading, Mass., 1975.
3. Ruth, S. S. and P. J. Kreutzer, "Data Compression for Large Business Files," *Datamation,* September 1972, pp. 62-66.
4. Shannon, C. E., "A Mathematical Theory of Communication," *Bell System Technical Journal 27,* July 1948, pp. 379-423, October 1948, pp. 623-656.
5. Baskett, F., K. M. Chandy, R. R. Muntz and F. G. Palacios, "Open, Closed and Mixed Networks of Queues with Different Classes of Customers," *JACM 22,* 2, April 1975, pp. 248-260.
6. Mathlab, *The Macsyma Reference Manual,* Version Seven, The Mathlab Group, Project MAC, Massachusetts Institute of Technology, September 1974.
7. Shannon, C. E., "Prediction and Entropy of Printed English," *Bell System Technical Journal 31,* January 1951, pp. 50-64.
8. Stonebraker, M. R. and E. Wong, *INGRES—A Relational Data Base System,* Electronics Research Laboratory Memorandum M-477, University of California, Berkeley, November 1974.
9. Smith, A. J., "A Performance Analysis of Multiple Channel Controllers," *Proc. 1st Annual SIGME Symposium on Measurement and Evaluation,* Palo Alto, Calif., February 1973, pp. 37-46.
10. Muntz, R. R. and J. W. Wong, "Efficient Computational Procedures for Closed Queueing Network Models," *Proc. Seventh Hawaii Conference on Computer and System Sciences,* January 1974, pp. 33-36.

The Federal Communications Commission and major policy matters affecting computer communication

by FRANK J. MARTIN, JR.
Sutherland, Asbill & Brennan
Washington, D.C.

ABSTRACT

This is a time of great change in FCC regulatory policies affecting computer communication. Over the next several months, the FCC will be called upon to consider and decide many issues which may well shape the nation's data communications for years to come.

This paper briefly reviews the important FCC cases which set the stage for today's policy issues. These include the *Specialized Common Carrier Decision,* the *Domestic Satellite Decision,* the *Interconnect Docket,* and the *Computer Inquiry.*

Then, the paper analyzes the recent FCC decisions and pending cases which are of special significance to the computer community. The AT&T DDS and HI-LO cases are discussed, as are the Satellite Business Systems proposal, the *Telenet v. Tymshare* complaint, and the AT&T DATASPEED 40 decision. The paper closes with a prediction that a new Computer Inquiry seems likely.

INTRODUCTION

This is an era of great upheaval in communications policy. In the *Specialized Common Carrier Decision,*[1] the Federal Communications Commission (FCC) made a deliberate decision to expand the domestic private line common carrier market beyond the two traditional carriers by authorizing new regulated carriers to enter the market and provide a range of specialized land-based data and other services in limited geographical areas. This was followed by the *Domestic Satellite Decision*[2] in which the FCC adopted a similar policy of multiple entry in the vitally important domestic satellite field. This new emphasis on competition in regulated long haul signal carriage has almost certainly increased the pace of innovation and the flexibility available to data communication users and to private line users generally. However, since AT&T dominates the entire communications field, and operates the enormously profitable monopoly interstate switched telephone services, in order to keep competition working the FCC has been forced to tackle the formidable, perhaps impossible, task of trying to assure that AT&T will not cross-subsidize its competitive services by use of monopoly profits, or otherwise use its monopoly control over necessary facilities to drive out competitors. Moreover, recent FCC actions having the effect of permitting IBM, the giant of the data processing industry, to enter the domestic satellite field as part owner of Satellite Business System,[3] means that the FCC has also taken on the task of assuring that IBM will not use its dominance in the closely related data processing field to gain competitive advantages in the communications field.

In the late sixties the FCC began to question and to strike down tariff restrictions designed to perpetuate carrier dominance in the incidental communications service and equipment fields. Thus, in the so-called *Interconnect Docket* (Docket 19528),[4] the FCC has recently striken most of the former tariff restrictions against use of customer-provided data and ancillary terminals. The FCC also eliminated the rest of these restrictions still in effect as to certain voice terminals and equipment.[5].

Although in its *Computer Inquiry*[6] and in subsequent authorizations of suppliers of so-called value-added networks as regulated common carriers,[7] the FCC seemed committed to full support of the traditional carrier restrictions against non-common carrier customer resale of communications circuits obtained from carriers, there are now some indications that in its decision in the *Resale and Shared Use* Proceeding,[8] expected sometime in early 1976, the FCC may relax or eliminate these restrictions altogether so as to permit resale and sharing on an unregulated basis. This could involve free entry of new communications middlemen, and actual detariffing of now regulated value added carriers and others, with resulting freer operation of competitive forces on pricing and the available range of customer choices.

There are also certain pending matters which focus on possible inadequacies of the FCC's Computer Rules and which will in all likelihood require some changes in those rules. In its *Computer Inquiry,* the FCC relied heavily on a distinction between message switching and data processing to distinguish communications services, which regulated carriers will be permitted to provide, from data processing services, which carriers are prohibited from providing except through separately incorporated subsidiaries. In other words, in the *Computer Inquiry,* the focus was on services and not on equipment and terminals. But now the carriers are offering PBX's (Private Branch Exchanges) which incorporate some of the systems architecture and perform some functions of computers. In addition, AT&T in particular is seeking to offer intelligent terminals pursuant to its tariffs. Are such items communications or data processing devices? Is the offering of such devices by carriers the equivalent of offering data processing services? (The FCC ruling on AT&T's DATA-SPEED 40 terminal will be discussed below.) Because of the inherent danger of cross-subsidy and interconnection restrictions by carriers, these are important questions which may affect competition in the terminal and equipment markets. In addition, the FCC is now focusing more precisely on some definitional questions raised in the *Computer Inquiry* as to what constitutes integral and incidental data processing or communications functions in a hybrid offering.

THE NEW SPECIALIZED AND DOMESTIC SATELLITE COMMON CARRIERS

Background: specialized carriers

In 1969 the FCC authorized Microwave Communications, Inc. ("MCI"), to provide point-to-point voice and data communications as a common carrier over microwave facilities between Chicago and St. Louis.[9] AT&T at first refused to provide local loops or local distribution facilities to MCI, claiming that such interconnection would harm AT&T's own communications network. However, the Commission held that AT&T did not prove that harm would result and AT&T was ordered to provide local distribution facilities for its new competitor.[10]

The MCI grant prompted an influx of applications by others seeking to provide specialized communications services in competition with the established common carriers. Instead of deciding each case individually the FCC initiated a rulemaking proceeding in Docket 18920.[11] Comments on all issues were received, including the need for more diverse and specialized data communications services, the impact of competition on the services and rates of established carriers, and the problem of local distribution, etc. Finally, in May of 1971 in its First Report and Order in Docket 18920, the FCC adopted a general policy favoring multiple entry of new specialized carriers. It stood by this decision on reconsideration, and its policy was sustained on appeal by the Ninth Circuit Court of Appeals.

In the *Specialized Common Carrier Decision,* and related new authorizations, the FCC thus established a policy that suppliers of domestic private line services should be expanded beyond the (then) existing nationwide carriers to include new carriers operating in limited geographical areas and offering a range of new specialized services including facilities for higher speed and more reliable data transmission at lower costs. In so doing the FCC stipulated that it would not provide a protective umbrella for the new carriers but that competition would rule the day. Existing carriers were to be allowed to depart from nationwide rate averaging where justified by costs or competitive necessity, and to take full advantage of any real economies of scale resulting from their overall operations, however, existing carriers were not to be permitted, through exercise of monopoly power, to withhold local distribution or other essential facilities from their new competitors, nor were they to be permitted to subsidize their competitive offerings by use of profits from monopoly services to drive their new competitors from the market.

Quality and reliability inquiry

The *Specialized Common Carrier* case, Docket 18920, is still in progress for determination of the measure of protection subscribers and users should be given respecting the quality and reliability of data communications services. The FCC tentatively decided against prescribing minimum standards of technical performance, but has proposed and sought comments on requiring carriers to specify, in standard terminology, the known quality and reliability supposedly available in particular offerings, and to make refunds where these standards are not met. Also, the FCC has suggested that carriers might be urged to publish periodic reports as to quality and reliability achieved, complaints received and refunds made.

In May of 1975, at the instance of one of the new specialized carriers, Data Transmissions, Inc. "Datran"), the FCC clarified the scope of Docket 18920 to include a definition of the scope of specialized services contemplated by the specialized common carrier authorizations.[12] The enormous importance of this is obvious, since it involves the scope of direct carrier competition to be allowed.

Satellite carriers

In the *Domestic Satellite Decision,*[13] the FCC similarly determined that the benefits of satellite technology could best be realized domestically by allowing multiple entry and competition, and by imposing on AT&T, the dominant domestic carrier, and Communications Satellite Corporation ("Comsat"), the international satellite carrier, certain requirements designed to assure that their other important services would not be burdened, and that their dominance in other communications fields would not be used for unfair competitive leverage. Last year, the FCC in essence approved entry in the domestic satellite field by IBM and Comsat General (a subsidiary of Comsat), who have recently announced that they, together with Aetna Life & Casualty Co. will seek FCC authority to establish a $250 million single satellite network for voice, data and image communications, to begin operations in 1979. Customers will access the new satellite network, which will be known as Satellite Business Systems, through 16 to 23 foot dish antennas mounted on their rooftops or elsewhere near their terminal sites, thus eliminating the need for interconnection with other carriers for local distribution or local loops.[14] Thus, the FCC has now taken on the task of supervising fair competition in data communications by the giant of the data processing industry, in addition to the formidable task of policing fair competition by the giant of the communications industry, AT&T.

The major issues.

The task of policing newly authorized competition has to date involved the two major problem areas anticipated by the FCC in the *Specialized Common Carrier Decision,* namely, (1) the problem of local distribution for the new competitors who as a practical matter cannot economically duplicate existing local distribution facilities controlled by AT&T and other telephone companies, and (2) the enormously complex problem of distinguishing between improper cross-subsidy by AT&T of its competitive services with monopoly profits, which was to be prohibited, and merely allowing AT&T to take full and fair advantage of real cost economies of scale, which the FCC promised it would allow.

Local facilities.

The matter of possible anticompetitive withholding of local distribution facilities arose in the Summer of 1973, when MCI advised the FCC that AT&T had refused to interconnect with MCI to provide local distribution through local telephone exchanges, even though AT&T did provide such exchange service in connection with its own competing private line services. The Chief, FCC Common Carrier Bureau advised by letter that under *MCI* and the *Specialized Common Carrier* decisions AT&T was obligated to provide local distribution through its affiliated company telephone exchanges. AT&T answered that it had no intention of participating in a "through service" with MCI or any other private line carrier, and that it would connect to the local exchanges only if state regulatory commissions authorized such interconnection. In a letter order, dated October 4, 1973, the Commission Chairman confirmed the earlier advice of the Common Carrier Bureau Chief, and stated that AT&T should submit its tariffs to the FCC only, and not to state regulatory commissions, since the latter had no jurisdiction of the matter. MCI then requested further clarification because despite the above, some AT&T affiliated companies declined to provide local interconnection for foreign exchange service (FX) or common control switching arrangements (CCSA) offered by the specialized carriers. The Common Carrier Bureau Chief again wrote to AT&T stating that FX, CCSA, and related services were covered by the Commission's October 4 letter order. Court contests were begun,[15] and in December of 1973 the FCC initiated Docket 19896,[16] ordering AT&T to show cause why it was not providing local distribution to the new specialized carriers. In April 1974, the Commission issued its decision in Docket 19896,[17] confirming that interconnection for local distribution had to be provided on a nondiscriminatory basis by AT&T, and this decision was sustained on review by the Third Circuit.[18] In consequence of this, and further tariff proceedings in Docket 20099 which followed, the parties eventually reached agreement on interconnection tariff provisions which the FCC accepted in May, 1975.[19]

The scope of state authority.

The question of the FCC's exclusive authority, as against state regulatory authorities, which was raised by AT&T in the course of its fight to withhold local distribution service from its new competitors, may not be entirely settled, however. The question is now pending review by the U. S. Court of Appeals for the District of Columbia in an appeal from an FCC order.[20] In that case, a petition for declaratory judgment was filed with the FCC by Southern Pacific Communications Company ("SPCC"), one of the new specialized carriers. SPCC claimed that the FCC's decision in Docket 19896, discussed above, entitled it to an order directing Pacific Telephone and Telegraph Co. and Southwestern Bell Telephone Co. to provide local distribution for certain of SPCC's FX and CCSA services, even though SPCC's facilities in question were located wholly within one state. In October 1975, the FCC ruled that it had exclusive jurisdiction to order interconnection since SPCC's facilities, though located entirely within one state, are nevertheless used though in a majority of instances as links in interstate communications. The FCC rejected the telephone companies' contention that interconnection to the local telephone exchanges was within the control of state utility commissions.[21]

Cross-subsidization

The vexatious problem of cross-subsidy has been under study by the FCC for at least 15 years. Recent developments have only resulted in greater urgency for a solution to an old problem. In *Nader v. FCC,* the United States Court of Appeals for the D.C. Circuit severely criticized the FCC's obvious inability to decide cross-subsidy and other important regulatory questions within a reasonable time.[22] In response to the court's order, the FCC was compelled to submit a schedule for decision of the cross subsidy issue in the *Private Line Case* (Docket 18128), and Phase II of Docket 19129 (AT&T Rate Case).

In compliance with that schedule, the Chief of the Common Carrier Bureau on January 19, 1976, issued a recommended decision which, if adopted by the Commission, will perhaps be the greatest setback ever experienced by AT&T before the FCC. In effect, the Bureau Chief held that AT&T has in fact been improperly subsidizing its competitive services with profits from its monopoly MTS and WTS services. For example, in August 1974 AT&T's rate of return on MTS and WATS were found to be 8.9%, and 12.3% respectively, whereas its rate of return in private line telephone and telegraph services were found to be 5.5% and −0.3%, respectively. AT&T's overall interstate rate of return was found to be 8.7%. Thus, in summary, AT&T's competitive service earnings were found to be significantly below its system-wide rate of return of 8.7%, while its monopoly service earnings were found to be significantly in excess of that system-wide rate of return.

The Bureau Chief rejected Bell's contention that fully distributed costs (FDC), the traditional cost allocation method used in rate making, should not be used. He refused to adopt AT&T's proposed "long run incremental cost" method (LRIC), as a substitute for fully distributed costs. Under LRIC, as advocated by AT&T, rates for competitive services would be based on management forecasts of future demand, technological developments, incremental investments and expenses, and competitive developments, while rates for monopoly services would be set at a high enough level to cover all costs, including historical costs, that were not easily identifiable or attributable to specific service offerings. The Bureau Chief did not reject altogether the concept of marginal cost pricing, i.e., pricing based on marginal, as opposed to full historical costs of providing a service, and suggested that Bell should look into further use of this concept in such things as peak/off-peak pricing of monopoly services.

The Bureau Chief recommended that AT&T be ordered to conduct a new comprehensive fully-distributed cost study for a recent one year test period, and to file tariffs which assure the minimum allowable overall rate of return in all categories of services. In early February of this year, in Docket 20376, AT&T's allowed rate of return was again increased from the 8.5% level established in 1972 in Phase I of Docket 19129, to 9.5%.[23]

Exceptions to the recommendations of the Chief of the Common Carrier Bureau were due March 19. Replies are due April 19. Following that the Commission has promised the Court of Appeals to issue its final decision in the *Private Line Case* by August 2, 1976.

In Phase II of Docket 19129, the Commission staff early this year filed proposed findings recommending that AT&T be required to divest Western Electric, and that many reductions be required in its claimed rate base. An initial decision by the Administrative Law Judge is, in response to the Court of Appeals requirement, scheduled for September 15, 1976, and final FCC action is scheduled for April, 1977.

Rate design issues.

But AT&T's reaction to the new competitors has not been limited to the struggle over interconnection to its own facilities, or defending itself against charges of cross-subsidy in preexisting competitive services. In November 1973, AT&T filed its so-called Hi-Lo tariff for its voice grade private line services (Series 2000/3000). By this filing AT&T sought to depart from its tradition of nationwide rate averaging and, in order to compete more effectively with the new specialized carriers, reduced rates on high-density routes, and established separate higher rates for short haul services of 25 miles or less. After an exchange of pleadings challenging and defending the Hi-Lo concept, the FCC designated the case for hearing[24] as to whether the Hi-Lo rates were based on cost savings or on competitive necessity or some other lawful consideration. The rates went into effect, subject to an accounting order and possible refund, on June 13, 1974.

In an Interim Decision issued September 18, 1975,[25] the FCC found that the Hi-Lo concept was reasonable, but that AT&T had not carried its burden of proof as to the reasonableness of the new rates, by showing that the criteria used for designating high-density, low-density and short-haul routes actually reflected network operations and actual costs. It accordingly remanded the case for further hearings on these matters.

But in January, 1976, in a surprising volteface, the FCC ruled that the Hi-Lo rates were illegal.[26] It noted that several parties, including MCI, had complained that AT&T's failure to carry the burden of proving claimed cost justification was grounds for outright rejection of the Hi-Lo tariff, and that in view of AT&T's persistent refusal to produce cost data, etc. in response to interrogatories, it would be a denial of due process to give AT&T a second chance at a further hearing. The FCC said on reconsideration that it did appear that AT&T did not have, and could not develop from records it had maintained, the cost and other data that would be necessary for possible justification of the Hi-Lo tariff. It accordingly decided to strike the tariffs as unlawful rather than allow them to remain in effect while AT&T developed extensive new studies and data in a further effort to justify them. AT&T was given 90 days from publication of the FCC's January decision in the Federal Register to file new tariffs, and 60 additional days to complete the cost justification data specified in the September Interim Decision. The FCC did not order any refund under the accounting order because it concluded that it did not have enough data to prescribe appropriate rates, and because the Hi-Lo tariff had lowered some rates while raising others with the result that most customers did not suffer significant net increase.

AT&T has also responded to the challenge of the new data oriented services of the specialized carriers by its ambitious Dataphone Digital Service (DDS) offering. In early 1973, it sought authority to supplement its existing microwave facilities between New York, Philadelphia, Chicago, Boston and Washington, D. C. to permit transmission of a digital bit stream, ultimately at 1.544 megabits per second, in an unused portion of the baseband on 4 to 6 GHz microwave radio channels. It was recognized that further approval would have to be obtained to before commercial service began under DDS, but even this first foot-in-the-door was vigorously opposed by Datran, MCI, SPCC, and other specialized carriers. The White House, through its Office of Telecommunications Policy (OTP), urged that any approval of DDS should be conditioned on strict requirements prohibiting cross-subsidy or other anticompetitive practices.

The FCC did not want to slow down provision of a needed service, and decided that the issues relating to competitive impact could be better addressed when commercial operating authority was sought. The FCC believed that by that time it would have concluded other important rate-making policy cases such as the *Private Line Case* (Docket 18128) involving cross-subsidy. In March 1974, AT&T filed its tariff to begin DDS commercial service, and offered such service initially between the five cities mentioned above, with expansion expected to 19 additional cities by late 1974. The service offered was two-way simultaneous digital transmission at 2.4, 4.8, 9.6 and 5.6 kilobits per second. The tariff rates were considerably below Datran's rates, and Datran and others argued that AT&T was obviously using its monopoly telephone service profits to subsidize its competitive digital data services. In December 1974, the FCC allowed the tariff to become effective on an experimental basis,[27] but ruled that the low rates would be applicable only in the five cities pending the outcome of an investigation into the lawfulness of those rates which it initiated as Docket 20288. The rates applicable in the additional 19 cities were to be the same as the higher private line rates for 12 months or until the investigation was concluded, whichever occurred earlier. The investigation was to include the cross-subsidy issue and also whether DDS was a class of service separate and distinct from AT&T's higher priced private line services.

The proceedings in Docket 20288 have been characterized by continuing disputes as to the nature and amount of cost data that AT&T should supply in response to the FCC hearing order and interrogatories filed by Datran and the separated trial staff. Proposed Findings were due to be filed in the case on March 15, 1976, and an Initial Decision is expected sometime later this year.

Recently AT&T filed tariff revisions to extend the low DDS rates to the 19 cities as to which higher rates applied for one year under the FCC's December 1974 designation order. The Independent Data Communications Manufacturers Association (IDCMA) and Datran opposed this, asserting that the experimental period had shown an adverse effect on competition and possible predatory pricing by AT&T. The Commission, however, refused to suspend or reject the effectiveness of the low rates in the 19 additional cities.[28] Then the FCC staff asked for reconsideration of this ruling on the ground that AT&T had admitted at a conference that it had not kept and could not produce cost data on DDS to show that the low DDS rates were cost justified. The staff contended that this showed that the tariff filing was unlawful. The Chief of the Common Carrier Bureau declined to reject the filing, but intimated that AT&T's failure to produce cost figures might well cause the DDS rates to be ultimately held illegal.[29] In view of the FCC's action this year in the Hi-Lo case, this does indeed seem quite possible.

CUSTOMER-PROVIDED TERMINAL EQUIPMENT

In June of 1972 the FCC began the so-called *Interconnect Proceeding,* Docket 19528. To assist it in making its determinations, the FCC established a federal/state Joint Advisory Board pursuant to § 410 of the Communications Act.[30]

After comments and reports on specific questions were received,[31] the Joint Board recommended, as an alternative to carrier-supplied connecting arrangements at customer expense, a type acceptance certification program to be administered by the FCC. The Joint Board recommended that the certification program apply to data and "ancillary" devices, including extension telephones, but not to other voice terminal equipment such as private branch exchanges (PBX's), key telephone systems, and main station and coin-operated telephones. FCC certification would, under the Joint Board's recommendation, be based on the applicant's submission of required test data. Also, each device would have affixed to it installation, maintenance and operating instructions. Standard plugs and jacks and other simple connecting devices were to be provided for in carrier tariffs.

In the fall of 1975, the FCC adopted a certification program similar to that recommended by the Joint Board.[32] In so doing, it noted that it had been seven years since *Carterphone* and that the carriers had never come forward with any justification either for the original tariff restrictions, or for the carrier supplied connecting arrangements subsequently provided for in the temporary tariffs.

The FCC ruled that its new certification program would adequately protect the telephone network from harm, i.e., from electrical hazards to personnel and equipment, and from degradation of service to persons other than the user of a particular device. Instead of requiring type approval of entire terminal devices, however, applicants will be permitted to register only connecting circuitry within any particular device. This will avoid problems of proprietory information disclosure raised by IBM and others. The new certification program, which applies to carrier-provided, as well as to customer-provided, data and "ancillary" equipment, including extension telephones, but not PBX's, key telephone systems and main station and coin operated telephones, was to go into effect April 1 of this year. However, the FCC has recently postponed this to May 1, 1976. In its recent ruling on reconsideration of the November decision,[33] the FCC also declined to exempt carrier-supplied terminal equipment from the new program and in general confirmed its original ruling.

The FCC followed the Joint Board's lead in leaving voice telephone equipment (other than extension telephones) subject to current tariff requirements for carrier-supplied connecting arrangements because, it said, the parties may not have addressed themselves specifically to this aspect. It did indicate strongly, however, that it saw no technical harm problems and proposed in the near future to issue further rule changes to include PBX's, key telephone systems, main station telephones, and coin telephones in the new certification program. Such action was, in fact, taken on March 18, 1976.[34]

AT&T and other carriers as well as the National Association of Regulatory Utilities Commissioners (NARUC), in petitions for reconsideration and stay of the new program, objected strongly that AT&T's "protective module" plan should have been adopted and argued that the FCC's plan is so deficient technically and otherwise that it cannot possibly work. They also argued that it was inappropriate and prejudicial for the FCC to allow unlimited interconnection of customer-supplied equipment prior to a determination, in the Economic Impact Inquiry, Docket No. 20003,[35] of the effect such interconnection will have on monopoly telephone services and rates. The Joint Board itself has recently voted to recommend this same course.[36]

In Docket 20003, the Commission is also considering the economic impact on the switched telephone network, and local telephone companies as well, of its recent authorizations of the new specialized and domestic satellite carriers, and the extent to which cross-subsidy of competitive offerings by monopoly carriers might be prevented by requiring them to make such offerings through subsidiary companies keeping separate books, etc.

There is perhaps an outside chance that the interconnect battle could be decided in a case now pending in the United States Court of Appeals for the Fourth Circuit.[37] The *Telerent Leasing* case, which was argued before the Fourth Circuit in September of last year, and at this writing has not been decided, involves an appeal from an FCC ruling in early 1974[38] that it alone has jurisdiction over the interconnection of customer-provided terminal equipment, even though such connection is to local telephone exchanges regulated by state utilities commissions. The effect of the Commission's ruling was to cancel a North Carolina Utilities Commission regulation prohibiting interconnection of "foreign attachments" of any kind. Opponents of the registration program contend that the FCC's rulings intrude unlawfully on local exchange jurisdiction given to state utility commissions by Section 221(b) by the Communications Act. However, the FCC ruled in *Telerent Leasing,* and contends before the Fourth Circuit Court of Appeals, that the interstate telephone service, over which it clearly has jurisdiction, is not and cannot possibly be provided except over local exchange facilities, and that it is evident that its exclusive jurisdiction over interstate communications must therefore extend to interconnection of terminal equipment to such facilities.

PROBLEMS RELATING TO THE DISTINCTION BETWEEN DATA PROCESSING AND COMMUNICATIONS

Background

In its *Computer Inquiry*[39] the FCC first explored the increasingly blurred relationship between communications and computers. It decided that whether or not computers were engaged in communications depended upon the use to which the computers were put. It recognized that computers perform communications functions—namely, message and circuit switching—and concluded that when doing so they are subject to regulation as a communications service. But the FCC seemed to say that everything else computers do including, inter alia, storing, retrieving, sorting, merging, and calculating according to programmed instructions is essentially data processing, which should not be regulated by the FCC. Because of the intermingling of communications with data processing in many remote access data processing offerings, and because such intermingling could lead to unfair competition in the data processing field by carriers who might resort to cross-subsidy of their data processing offerings, or restrictive interconnection practices as to data processing offerings of others, it was decided that the FCC would not permit regulated carriers to offer essentially data processing services, except through separate subsidiary corporations keeping separate books. Where data processing and communications (including computer controlled message or circuit switching) are both involved in a service (i.e., where there is a "hybrid" service), then whether the service is deemed hybrid *communications,* such as may be provided by a common carrier, or require regulation if provided by others, is a question of the primary purpose of the offering. If the primary purpose is found to be to serve signal carriage and/or message-switching requirements of a customer, and data processing is an integral but incidental part of a package offering, the offering is deemed a hybrid *communications* offering and may be offered by carriers and is subject to regulation if offered by others. On the other hand, if the primary purpose is to provide data processing services, and computerized message or circuit switching (i.e., communications) is an integral but incidental part of a package offering, the offering is deemed to be hybrid *data processing* service and may not be offered by carriers and is not subject to regulation.

To enforce its computer rules, the FCC decided to rely basically on the regulated carriers from whom the communications component of any hybrid offering to be made by a non-carrier would necessarily have to be acquired. Since the carrier tariffs prohibit customer resale of communications services, and hybrid communications offerings would involve such resale, the carriers themselves were expected simply to refuse to supply or to continue to supply communications to anyone planning to resell the same as a component of a hybrid communications offering, unless the customer in question had been authorized by the FCC to offer such service as a communications common carrier.

In recent years pursuant to the principles outlined in the *Computer Inquiry,* the Commission has authorized three so-called value added carriers to offer "packet switched" data service employing lines leased from other carriers combined with their own computers and software to transmit small groups (packets) of digitized data using store and forward methods to take advantage of the best available path through the network.[40]

Telenet v. Tymshare.

In early August of 1975, the tariffs of one of these newly authorized value added carriers, Telenet Communications Corporation, became effective. Five weeks later Telenet filed a complaint against Tymshare, Inc., a time-sharing concern, alleging that Tymshare's Tymnet data communications network, which also utilizes store and forward switching computers and leased communication channels, is physically and functionally separate from Tymshare's data processing computers which are connected to Tymnet. It charged that Tymshare, Inc. is in reality engaged in two separate profit making businesses: first, a hybrid *data processing* service using the Tymnet network as a means of communications between customer terminals and Tymshare's host computers; and, second, a hybrid *communications* service whereby it offers use of Tymnet as a means of communications between customers' terminals and the customers' own host computers. Tymshare, Inc. has answered this complaint alleging that it is not purely coincidental that Telenet filed only a few weeks after its tariffs went into effect. It suggests that Telenet really hopes to use FCC procedures to advance what it conceives to be its own competitive interests, and points out that the question whether, and to what extent, value-added carriers and others may be permitted to lease and resell or enter into joint user arrangements so as to provide value added services to others is presently under consideration and the Commission's inquiry in the *Resale and Shared Use Proceeding,* Docket No. 20097.[41] Tymshare argues that its Tymnet system is operated pursuant to the so-called joint user provisions of relevant tariffs, i.e., Tymshare leases lines from AT&T and other carriers for internal telecommunications and makes unused capacity available to others under joint user tariff provisions. In general, these require that the customer (Tymshare) have its own communications needs over and above those arising from management of the joint use arrangement, and that the customer and the joint users share the cost of the common carrier service by each paying part of the rates to the common carrier. Tymshare further states that the FCC in its order in initiating Docket No. 20097, expressly recognized that sharing arrangements could involve a complex computer switched network.

Tymshare also asserts that neither it nor even Telenet is in reality, or ought to be, considered a communications common carrier. This contention by Tymshare is squarely directed at the wisdom of the FCC's readiness to characterize value added services as communications because they involve circuit and message switching. At this writing, the FCC's staff has queried the carriers who are supplying communications circuits to Tymshare for their views as to whether Tymshare is indeed in full compliance with their tariff provisions prohibiting resale except under bona fide sharing arrangements. In short, for the time being the staff is following the FCC's suggestion in the *Computer Inquiry* that the carriers should first make the difficult decision whether a hybrid communications service is involved. But it seems inevitable that the FCC will ultimately have to decide this tough question as well, and the Tymshare case bears close watching.[42]

Other FCC cases

A similar situation where a carrier is asserting that a purported data processing service is in reality a communication service, involves ITT-Worldcom's pending challenge of the Telepost computerized message service being provided by TII Corporation. A customer signals Telepost that certain prewritten messages are to be sent to persons listed on various pre-established customer lists and the Telepost computer automatically interprets these signals and sends out the desired messages to the desired recipients via MAILGRAM. ITT-Worldcom claims in a petition in the proceeding that this service is a hybrid communications service which should be subject to regulation.

New developments in the service business are also putting to the test the *Computer Inquiry* definition of hybrid data processing and hybrid communications. Western Union has recently petitioned to offer a collateral processing service in conjunction with its SICOM message service for the securities industry. Under the proposed collateral service, Western Union's computers would perform order matching based on information gleaned from the handling of buy and sell messages. Western Union contends that this is a hybrid data communications service, since the data processing functions of order matching are both incidental to an integral with the SICOM message service. But this interpretation is being challenged by CBEMA and ADAPSO. The latter say that the proposed service is a data processing service that is neither incidental nor integral to Western Union's SICOM communications service. The Commission is being called upon to interpret its *Computer Inquiry* rules to determine whether Western Union will be permitted to offer this service.

The pattern of competitive response by AT&T, followed by charges and inquiries whether the competitive response is in fact predatory and subsidized by monopoly profits, has held true in the terminal equipment area as well as in the specialized common carrier services area and others. Such matters are now at issue in the AT&T Data Modem Rate Investigation, Docket 19419, which is now in hearing before an FCC Administrative Law Judge.

AT&T DATASPEED 40 Ruling

Another instance of such a competitive response by AT&T, which also illustrates the importance of the *Computer Inquiry* to the terminal equipment field, as well as in the service field, is involved in AT&T's recent tariff filings relating to Dataspeed 40. In these filings, AT&T sought to provide, subject to FCC regulation, intelligent remote access terminals with cathode ray video display. These terminals were designed to be for use by customers in connection with AT&T's Dataphone Digital Service, and to be competitive with new terminals manufactured by unregulated computer and data processing manufacturers including IBM. In petitions to reject or suspend these filings, IBM, CBEMA and others contended that AT&T's Dataspeed 40 terminals were plainly data processing equipment since they are in direct competition with IBM terminals which are not and should not be subject to tariffs at all. They argued further that AT&T is prohibited from competing in the data processing field in this manner by a 1956 antitrust consent decree which limits AT&T to common carrier communication services and services "incidental" thereto.[43] They argued that AT&T's filings were unlawful since they did not demonstrate conclusively that the new terminal was not being cross-subsidized by AT&T. They suggested further that if AT&T is, nevertheless to be permitted to offer data processing equipment, a revision of the 1956 antitrust decree would be required, and AT&T would have to offer the equipment on an untariffed basis through a separate subsidiary in accordance with the FCC's ruling in its *Computer Inquiry* respecting data processing services offered by regulated carriers.

The FCC's staff recently rejected the Dataspeed 40 offering, concluding in essence that it amounted to a data processing, not a communications service.[44] AT&T had argued the new terminals were merely evolutionary improvements of teletypewriter terminals which it has provided for many years as communications devices. It argued that if a computer—which consists of input, output, arithmetic and logic, memory and control units—is used to execute a program that involves data processing, the data processing takes place only in the arithmetic and logic units of the computer. The input and output devices perform no data processing since they do not operate on information to increase its worth to the user through changing its inherent informational content; instead the input and output devices merely permit outside entities to converse with the central processing unit and are required because of inherent limitations of the central processing unit. AT&T argued, therefore, that the Dataspeed 40 terminal performs the same communications function as that performed by telephones when two human beings converse remotely.

The staff rejected AT&T's arguments, however stating that it was clear that the primary function, design, and marketing of the Dataspeed 40 terminal was as an integral part of a data processing service involving the programmed interconnection of the terminal device and a central computer processing and/or storage unit. The FCC staff pointed out, and laid emphasis on the fact, that the terminals cannot communicate with themselves without the use of external data processing equipment.

If the staff's rejection of Dataspeed 40 is upheld, this bodes ill for AT&T's recently announced plans to offer, in the latter part of this decade, a new end-to-end, value added "communications processing" service designed to provide an alternative to the systems network architecture to be offered by IBM. Under this plan AT&T's computer controlled No. 4 ESS switch, located at the telephone company's central office, would function for users, on a shared basis, as a substitute for central processor front ends, multiplexors, concentrators, remote controllers and intelligent terminals. Users could bypass the profusion of separate systems for different on-line applications, and disparate outputs could be loaded into a single system capable of rearranging each set of bits without changing informational content, to meet requirements of receiving terminals. AT&T recently introduced "Dimension PBX" as the prototype for an on site controller which will be used for some applications of the new centralized communications processor.

AT&T will have to fight back contentions that its alleged "communications processing" service is in reality data processing, and a recent AT&T announcement gave a preview of what its position will be on this issue. It stated that data processing involves altering informational content of bits, something the centralized communications processing system will not do. Instead, AT&T said, the new system will perform network control, speed conversion, area control, terminal polling, message routing and rerouting (to effect priorities and avoid busy or down links), formatting, editing, and checking of input and output data. The announcement further explained that "communications processing" is but one of three elements of "data communications." The other two are transmission/switching/modulation/demodulation, and media conversion (e.g., transformation of bit stream to hard copy, cathode ray tube display (CRT display) or punched cards).

CONCLUSION

In a speech made by Walter R. Hinchman, Chief of the Common Carrier Bureau at the FCC, before the Computer Industry Association (CIA) in Washington, D. C. on February 25, 1976, Mr. Hinchman stated that problems posed by some of the cases discussed above are causing the FCC to have to focus closely on some of the ambiguities in its *Computer Inquiry* rules and, he said, might lead to a reexamination of these rules to make them more relevant in the context of changing conditions in 1976.[45]

It appears likely that a new Computer Inquiry will be launched. And this time it will be considered in a totally new environment—one with EFTS, privacy, and technology issues quite different from those dealt with previously.

REFERENCES

1. *In the Matter of Policies and Procedures for Consideration of Applications to Provide Specialized Common Carrier Service in the Domestic Public Point-to-Point Microwave Radio Service and Proposed Amendment to Parts 21, 43 and 61 of the Commission's Rules* (Docket 19820), *First Report and Order in Docket 18920,* 29 F.C.C. 2d 870 (1971); *Memorandum Opinion and Order on Reconsideration,* 31 F.C.C. 2d 1106 (1971); *affirmed, sub nom. Washington Utilities and Transportation Commission v. F.C.C.,* 513 F.2d 1142 (9th Cir.), *cert. denied,* 423 U.S. 836, 1976.
2. *In the Matter of Establishment of Domestic Communications—Satellite Facilities by Non-government Entities* (Docket 16495), *First Report and Order,* 22 F.C.C. 2d 86 (1970); *Notice of Proposed Rulemaking,* 22 F.C.C. 2d 810 (1970); *Memorandum Opinion and Order Inviting Comments on Staff Recommended Decision,* 34 F.C.C. 2d 1 (1972); *Second Report and Order;* 35 F.C.C. 2d 844 (1972); *Memorandum Opinion and Order on Reconsideration,* 38 F.C.C. 2d 665 (1972). *See also, AT&T Domestic Satellite Authorization,* 42 F.C.C. 2d 654, 1973.
3. *In the Matter of Petition for Approval of Changes in Corporate Structure of CML Satellite Corporation* (Docket 20221), *Memorandum Opinion and Order,* 51 F.C.C. 2d 14 (1975); *Memorandum Opinion and Order,* — F.C.C. 2d — (F.C.C. 75-986, released September 26, 1975); FCC Report No. I-204 (Notice of Title III Satellite and Overseas Service Applications Accepted for Filing) (Feb. 6, 1976). *See Wall Street Journal,* December 24, 1975, p. 40.
4. *In the Matter of Proposals for New or Revised Classes of Service of Interstate and Foreign Message Toll Telephone Service (MTS) and Wide Area Telephone Service (WATS)* (Docket 19528), *Notice of Inquiry and Proposed Rulemaking,* 35 F.C.C. 2d 539 (1972); *First Supplemental Notice of Inquiry,* 40 F.C.C. 2d 315 (1973); *First Report and Order,* 56 F.C.C. 2d 593 (F.C.C. 75-1248, released November 7, 1975); *On Reconsideration,* — F.C.C. 2d — (F.C.C. 76-134, released February 13, 1976).
5. *Id., Second Report and Order,* — F.C.C. 2d — (F.C.C. 76-242, released March 18, 1976).
6. *Regulatory and Policy Problems Presented by the Interdependence of Computer and Communications Services and Facilities* (Docket 16979), *Notice of Inquiry,* 7 F.C.C. 2d 11 (1966); *Tentative Decision,* 28 F.C.C. 2d 291 (1970); *First Decision and Order,* 28 F.C.C. 2d 557 (1972); *affirmed in part, GTE Service Corp. v. F.C.C.,* 474 F.2d 724 (2d Cir. 1973).
7. *Packet Communications, Inc.,* 43 F.C.C. 2d 992 (1973); *Telenet Communications Corp.,* 46 F.C.C. 2d 680 (1974); *Graphnet Systems, Inc.,* 44 F.C.C. 2d 800, 1974.
8. *Regulatory Policies Concerning Resale and Shared Use of Common Services and Facilities* (Docket 20097), *Notice of Inquiry and Proposed Rulemaking,* 47 F.C.C. 2d 644, 1974.
9. *Microwave Communications, Inc.,* 18 F.C.C. 2d 953, 1969.
10. *Microwave Communications, Inc.,* 21 F.C.C. 2d 190, 1970.
11. See Note 1, *supra.*
12. 52 F.C.C. 2d 1037 (F.C.C. 75-449, released May 2, 1975).
13. *See* Note 2, *supra.*
14. *See Datamation,* February 1976, pp. 114-118.
15. *See, e.g., MCI Communications Corp. v. American Telephone & Telegraph Co.,* 496 F.2d 214 (3d Cir. 1974).
16. *In the Matter of Bell System Tariff Offerings,* 44 F.C.C. 2d 245, *modified,* 44 F.C.C. 2d 914, 1974.
17. *In the Matter of Bell System Tariff Offerings,* 46 F.C.C. 2d 413, 1974.
18. *Bell Telephone Company of Pennsylvania v. F.C.C.,* 503 F.2d 1250 (3d Cir. 1974), *cert. denied,* 422 U.S. 1026, 1975.
19. *In the Matter of AT&T Offer of Facilities for Use by Other Common Carriers,* 47 F.C.C. 2d 660 (1973), *modified,* 49 F.C.C. 2d 729 (1974); *Acceptance of Settlement Agreement* 52 F.C.C. 2d 727 (F.C.C. 75-450, May 7, 1975); Civil Action No. —, F.C.C. Memo 53813, August 8, 1975.
20. U.S. Court of Appeals for the District of Columbia No. 75-2060. *See* F.C.C. Memo 59036, December 19, 1975.
21. — F.C.C. 2d — (F.C.C. 75-1146, released October 16, 1975).
22. *Nader v. F.C.C.,* 520 F.2d 182, 35 Pike and Fisher, Radio Regulation 2d 187, September 29, 1975.
23. *In the Matter of AT&T Charges for Interstate Telephone*

Service (Docket 20376) — F.C.C. 2d — (F.C.C. 76-100; released February 5, 1976).

24. *In the Matter of AT&T Charges, Regulations, Classification and Practices for Voice Grade Private Line Service* (*High Density—Low Density Rate Structure*) (Docket 19919), 44 F.C.C. 2d 697 (1974).
25. *Id.*, 55 F.C.C. 2d 224, 1975.
26. *Id., Decision and Memorandum Opinion and Order,* — F.C.C. 2d — (F.C.C. 76-30; released January 22, 1976). An appeal is pending before the U.S. Court of Appeals for the District of Columbia, Appeal No. 75-2059.
27. *In the Matter of AT&T Proposed Tariff F.C.C. No. 267, Offering a Dataphone Digital Service Between Five Cities* (Docket 20288) 50 F.C.C. 2d 501, 1974.
28. *Id.*, — F.C.C. 2d —, (F.C.C. 75-1341, released December 17, 1975).
29. *In the Matter of AT&T Revisions of Tariffs F.C.C. Nos. 260, 267 and 268, Memorandum Opinion and Order*, adopted January 26, 1976, released January 28, 1976 (F.C.C. Memo No. 60297).
30. *In the Matter of Proposals for New or Revised Classes of Interstate and Foreign Message Toll Telephone Service* (*MTS*) *and Wide Area Telephone Service* (*WATS*) (Docket 19528), 35 F.C.C. 2d 539, 1972.
31. *Id., First Supplemental Notice,* 40 F.C.C. 2d 315 (1973).
32. *Id., First Report and Order,* — F.C.C. 2d — (F.C.C. 75-1248, released November 7, 1975). This case is now pending on appeal to the Fourth Circuit, Case No. 76-1002.
33. *Id., On Reconsideration,* — F.C.C. 2d — (F.C.C. 76-134, released February 13, 1976).
34. See Note 5, *supra.*
35. *In the Matter of Economic Implications and Interrelationships Arising From Policies and Practices Relating to Customer Interconnection, etc.* (Docket 20003), *Notice of Inquiry,* 46 F.C.C. 2d 214, 1974.
36. Joint Board Recommendation of February, 1976 against inclusion of PBX's etc. in Certification Program.
37. *Telerent Leasing Corp. v. F.C.C.*, Case No. 74-1220 pending in the U.S. Court of Appeals for the Fourth Circuit.
38. *Telerent Leasing Corp.*, 45 F.C.C. 2d 204 (February 5, 1974).
39. See Note 6, *supra.*
40. See Note 7, *supra.*
41. See Note 8, *supra.*
42. *NARUC v. F.C.C.*, 525 F.2d 630, 644 (D.C. Cir. 1976) suggests that the F.C.C. has very limited discretion in deciding whether to confer "common carrier status" on a given entity.
43. *Western Electric Co., Inc. and American Telephone and Telegraph Co.*, (consent judgment), 13 Pike and Fisher, Radio Regulation 2143, 1956 Trade Cases, ¶ 71, 134 (D.C.N.J., Jan. 24, 1956).
44. *In re AT&T,* Transmittal No. 12449, Mimeo No. 61760, released March 3, 1976.
45. FCC release entitled, Remarks of Walter R. Hinchman, Chief, Common Carrier Bureau, Federal Communications Commission, Before the Computer Industry Association, "Current Developments in Common Carrier Communications," Washington, D.C., February 25, 1976.

A new communication protocol for accessing data networks—The international packet-mode interface

by A. RYBCZYNSKI
Bell Canada
Ottawa, Canada

B. WESSLER
Telenet Communications Corporation
Washington, D.C.

R. DESPRES
Administration Francaise de PTT
Rennes, France

and

J. WEDLAKE
United Kingdom Post Office
London, England

ABSTRACT

Public packet switching networks are at various stages of development around the world, notably in the U.S., Canada, France, the United Kingdom and Japan. The success of these networks is highly dependent on the use of an agreed-upon standard device-independent interface between the packet networks and the user devices operating in the packet-mode. This interface consists of far more than the data link control procedure (i.e., HDLC), which administers the physical transmission medium between the data terminal equipment (DTE) and the network. The specification of the packet-mode interface defines a set of conventions governing the manner in which DTEs establish, maintain and clear calls, format control information and data into packets and manage the flow of data for many calls over a single circuit to and from the packet network.

This paper describes the International Packet-Mode Interface, developed jointly by Telenet Communications Corp., the Trans-Canada Telephone System (TCTS), the United Kingdom Post Office and the French PTT. This interface has been designed to enable DTEs such as computers, programmable terminal controllers and intelligent terminals to gain access to public packet networks throughout the world. The present status of international standardization of this interface within the CCITT is also covered.

Standardization of the International Packet-Mode Interface is to the advantage of teleprocessing users, manufacturers of data processing and terminal equipment and common carriers.

INTRODUCTION

Telenet has been commercially available in the U. S. since August 1975; Datapac will start commercial operation in Canada next month; the French experimental RCP network started operation in 1975 and Transpac development has been contracted out and will start providing service in France in 1978; the Experimental Packet Switching Service of the United Kingdom Post Office is well into the evaluation stage; and other public data networks are being planned and developed in both Japan and the Nordic countries.

The fundamental technology used in all of these networks is packet switching. In packet switching, all user data is formed into discrete units called packets. In addition to the data to be transferred (typically part of a message), each packet includes a header specifying control functions and the destination to which the data is to be delivered. Packets are routed through the network on a store and forward basis and travel very rapidly and accurately through the network, experiencing only a fraction of a second delay from source to destination. Additionally, the network performs buffering functions so that the speed and format of the data sent into the network can be different from those of the data received at the destination.

The packet switching technology described briefly in the preceding paragraph is actually of little direct concern here, since the sophisticated routing, monitoring and error correction techniques internal to packet networks are invisible to users of the networks. (Readers are referred to a book edited by Chu[1] for a discussion of packet network design considerations). Rather, the user's primary interests are the characteristics of the service provided by a network and of the interface between him and the network.

The public data networks listed above are supporting packet-mode services based on the *virtual circuit* concept. A virtual circuit is a bi-directional association between a pair of DTEs over which all data transfer takes the form of packets. Transmission facilities are only assigned when data or control packets are actually being transferred. The virtual circuit concept permits a DTE (data terminal equipment) to establish concurrent communications paths to many other DTEs over a single physical access circuit (Figure 1). The high degree of sharing made possible by the use of virtual circuits enables communication savings to be passed on to the user. The virtual circuit concept minimizes the impact of new packet switching services on existing user systems.

Two types of interfaces may exist on any packet switching network. The first type may be called a device-dependent interface such as would be required at present for most hard-wired terminals (e.g., point-of-sale terminals, teletype machines). The second type may be called a device-independent interface which is applicable to most programmable devices (e.g., computers, programmable controllers, concentrators, intelligent terminals). Because of the large number of terminals that require device-dependent interfaces and because of the large incompatibilities that presently exist among these terminals, a large number of interface specifications are required. On the other hand, only a single specification of a device-independent interface is necessary.

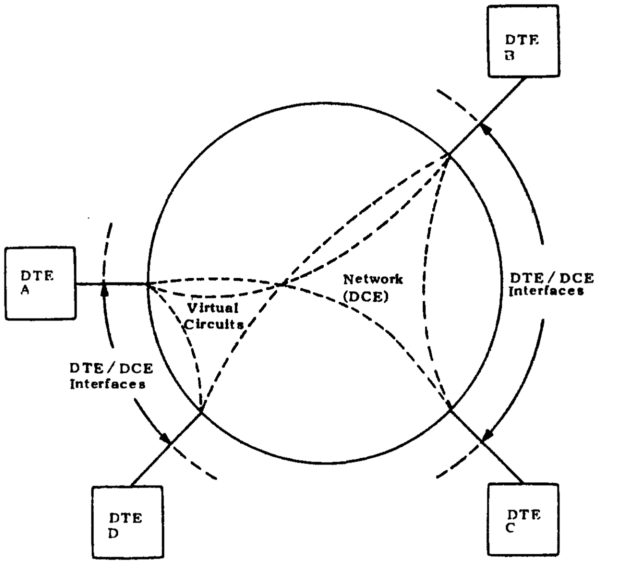


Figure 1—Use of virtual circuits

The International Telephone and Telegraph Consultative Committee (CCITT) in Geneva has recognized the need for these two types of interfaces and has put priority on the development of a recommendation for the second type of interface; that is, a device-independent interface between what it calls a packet-terminal and the packet switching network. A similar conclusion has been drawn by an Ad Hoc Group of U. S. ANSI Task Group X3S37 (Public Data Networks).[2]

Likewise, TCTS, the French PTT, Telenet, the Japanese NTT, and the United Kingdom Post Office have put high priority on the development and standardization of the International Packet-Mode Interface. The specification of the International Packet-Mode Interface was submitted as draft Recommendation X.25[3] to the CCITT Secretariat for consideration by the Study Group VII Plenary.

The purpose of specifying the International Packet-Mode Interface is to provide an efficient means by which a large set of characteristically different teleprocessing systems can gain access to the services and related technical and economic benefits of packet switching networks.

GENERAL DESCRIPTION

Interface requirements

The basic requirements imposed upon the architecture of the interface are introduced below:

1. The interface shall provide a full duplex transmission path between the DTE and the network.
2. It shall ensure the integrity and accuracy of the data transmitted between the DTE and the network.
3. It shall provide the DTE with switched and permanent virtual circuits.
4. It shall be capable of efficiently supporting concurrent communication between a packet mode DTE and numerous other DTEs over a single physical circuit to the network.
5. It shall allow both DTE and network to control the flow of data over the access circuit so that one does not overload the other.
6. It shall provide supervisory and control functions to administer calls satisfactorily.
7. It shall do all of above using existing standards wherever possible.

Interface characteristics

The International Packet-Mode Interface consists of three distinct levels of control procedures as illustrated in Figure 2:

1. the Physical Interface
2. the Frame Level Logical Interface
3. the Packet Level Logical Interface

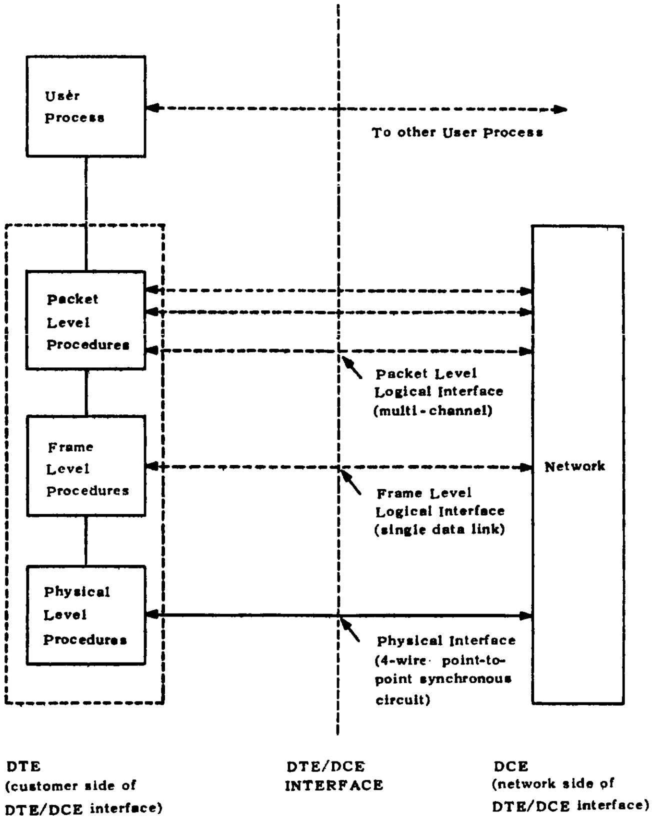
Each of these levels functions independently of the other levels, with the exception that failures at a level may affect the operation of higher levels.

The Physical Interface specifies the use of a duplex, point-to-point synchronous circuit, thus providing a physical transmission path between the DTE and the Network. It also specifies the use of an existing physical interface (i.e., EIA RS-232-C standard) between the DTE and a data set or modem. Therefore, no changes to the interface hardware of the DTE are required.

The Frame Level Logical Interface specifies the use of a data link control procedure which is compatible with the High-Level Data Link Control (HDLC) procedures being standardized by ISO and with the Advanced Data Communications Control Procedure (ADCCP) being standardized by U. S. ANSI. The Frame Level Interface uses the principles of a new ISO Class of Procedure for a point-to-point balanced system, whereby the DTE and the network node each have a primary and a secondary function. The Frame Level Interface is defined in terms of primary and secondary responsibilities, and may be thought of as two independent but complementary transmission paths superimposed on a single physical circuit. The use of this data link control procedure ensures that packets provided by the packet level and contained in HDLC information frames are accurately exchanged between the DTE and the Network. The functions performed by the Frame Level Interface are:

1. transfer of data in an efficient and timely fashion;
2. synchronizing the link to ensure that the receiver is in step with the transmitter;
3. detecting transmission errors and taking steps to recover from such errors;
4. identifying and reporting procedural errors to higher levels for recovery.

The major significance of the Frame Level Interface is that it provides the Packet Level Logical Interface with an error-free, variable delay link between the DTE and the Network.


User Process
To other User Process
Packet Level Procedures
Packet Level Logical Interface (multi-channel)
Frame Level Procedures
Frame Level Logical Interface (single data link)
Network
Physical Level Procedures
Physical Interface (4-wire point-to-point synchronous circuit)
DTE (customer side of DTE/DCE interface)
DTE/DCE INTERFACE
DCE (network side of DTE/DCE interface)

Note: Network is transparent to process-to-process communication.

DCE = Data Circuit-Terminating Equipment (e.g. data set)

Figure 2—International packet-mode interface architecture

The *Packet Level Logical Interface* is the highest level of the International Packet-Mode Interface and specifies the manner in which control information and user data are structured into packets. The control information including addressing information is contained in the packet header field and allows the network to identify the DTE for which the packet is destined. It also allows a single physical circuit to support a number of virtual circuits to numerous other DTEs concurrently.

The Packet Level Logical Interface is further described in the next section.

THE PACKET LEVEL LOGICAL INTERFACE

Multiplexing at the packet level

The Packet Level Logical Interface accommodates both permanent and switched virtual circuits. A permanent virtual circuit is a permanent association existing between two DTEs which is analogous to a point-to-point private line. Thus, it requires no call set up or call clearing action by the DTE. A switched virtual circuit is a temporary association between two DTEs and is initiated by a DTE sending a call request packet to the network. Call establishment and clearing is described in the next section.

In order to allow a DTE to establish concurrent virtual circuits with a number of DTEs over a single physical access circuit, the Packet Level Logical Interface employs packet-interleaved Statistical Multiplexing. This multiplexing technique is used to exploit the fact that a typical virtual circuit to a remote DTE may actually be carrying data for only a small percentage of the time. Each packet contains a logical channel number which identifies the packet with a switched or permanent virtual circuit for both directions of transmission. A packet that contains user data for example has a three octet header identifying it as a data packet and specifying its logical channel number as illustrated in Figure 3.

Call establishment and clearing

A signalling method is provided to allow a DTE to establish switched virtual circuits to other DTEs using logical channel numbers at each end to locally designate these switched virtual circuits.

A DTE initiates a call by sending a call request packet, Figure 4, to the Network. The call request packet includes the logical channel number chosen by the DTE to be used to identify all packets associated with that call. It also includes the network address of the called DTE. A facility field is present only when the DTE wishes to request an optional user facility (i.e., network feature) requiring some indication at call set up. Reverse charging is an example of such a facility. User data may follow the facility field and may contain any number of bits up to a maximum of 16 octets.

The calling DTE will receive a response indicating whether or not the called DTE accepts the call. When a switched virtual circuit cannot be established, the network will transfer clearing call progress signals to the DTE indicating the reason why the call was not established; call progress signals are listed in Table I. Either DTE may clear an established call with this information being conveyed to the opposite DTE.

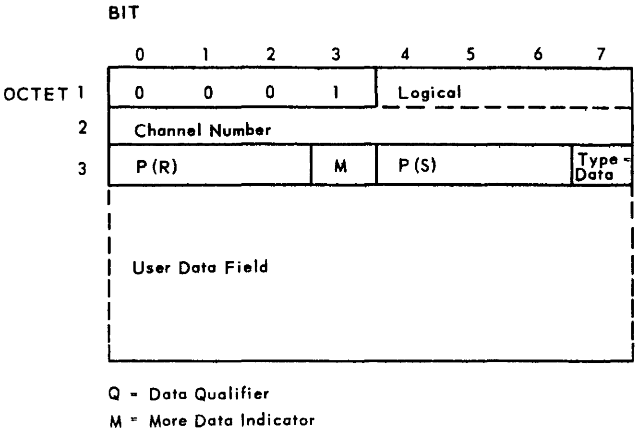
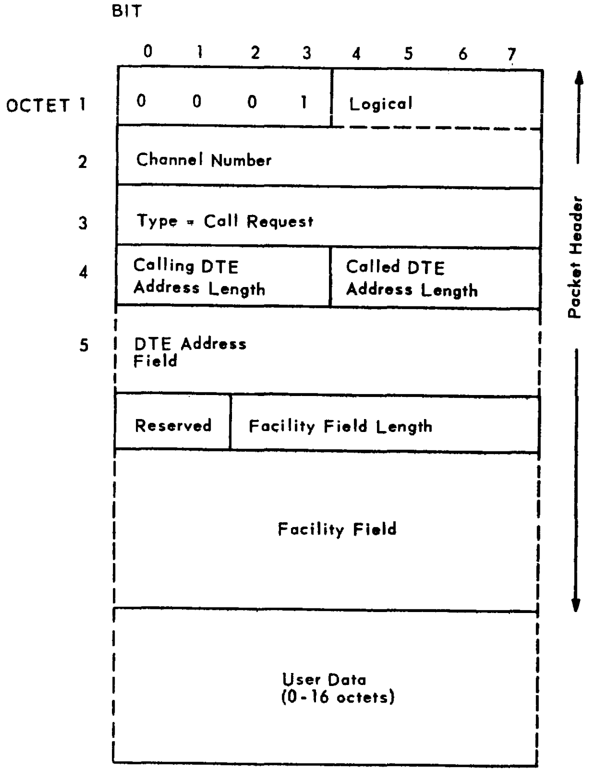


Figure 3—Data packet format



Bits of an octet are numbered 0 to 7, where bit 7 is the low order bit and is transmitted first. Octets of a packet are numbered consecutively starting from 1 and are transmitted in this order.

Every packet header has a 4-bit field which is effectively reserved for future use, a 12-bit logical channel number and an 8-bit field used for packet type information and control. This latter field is intentionally kept very similar to the control field of HDLC.

Figure 4—Call request packet format

Figure 5 is an illustration of call establishment, data transfer and call clearing.

Data transfer on a virtual circuit

Data packets, illustrated in Figure 2, can only be transferred on a virtual circuit after the virtual circuit has been established *and* flow control constraints are not violated. The third octet of the data packet header is identical to the control byte of HDLC information frames except that the poll/final bit is replaced by the More Data bit discussed later.

TABLE I—Clearing Call Progress Signals

Clearing Call Progress Signal	Explanation
Number Busy	The called number is fully engaged and cannot accept another call.
Number Refusing Collect Calls	The called DTE does not accept collect calls.
Network Congestion	Congestion conditions within the network temporarily prevent the requested virtual circuit from being established.
Invalid Call	Invalid Facility requested.
Access Barred	The calling DTE is not permitted to obtain the connection to the called number. Possible reason is incompatible closed user group.
Local Procedure Error	The call is cleared because of a local procedure error.
Remote Procedure Error	The call is cleared because of a remote procedure error.
Not obtainable	The called number is not assigned or is no longer assigned.
Out of Order	The called number is out of order. Possible reasons include (1) DTE not functioning; (2) Subscriber link not functioning; (3) Frame level not in operation.

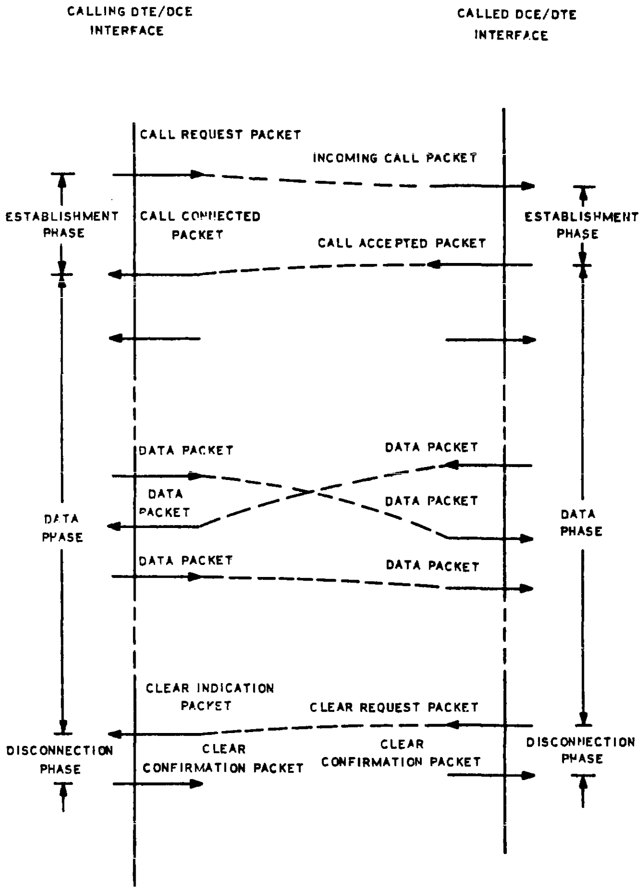


Figure 5—Illustration of call establishment, data transfer and call clearing

P(S) is the packet send sequence number of the packet. (Only data packets are numbered, modulo 8). The maximum number of sequentially numbered data packets that the DTE may be authorized to transmit, without further authorization from the network, may never exceed seven. The actual maximum value, called the window size W, is set for the virtual circuit either at subscription time or at call set up.

Each data packet also carries a packet receive sequence number P(R) which authorizes the transmission of W data packets on this virtual circuit starting with a send sequence number equal to the value of P(R). If the DTE or the network wishes to authorize the transmission of one or more data packets across the interface, but there is no data flow on a given virtual circuit in the reverse direction on which to piggyback this information, it can transmit a Receive Ready (RR) packet. Flow control based on the conveyance of P(R) numbers on a virtual circuit basis ensures that a sending DTE does not transmit data at an average rate which is greater than that at which the receiving DTE can accept that data.

The data field of a data packet to be transmitted on a virtual circuit may be any number of bits long up to some maximum value. The latter may be established independently at each end of a virtual circuit. Every network will support a maximum value of 128 octets. It may optionally support other values, possible values being 16, 32, 64, 255, 256, 512 and 1024. The governing principle is that a virtual circuit is used for the transfer of streams of user bits, where packet size may be chosen in such a way as to locally optimize: access line performance, cost, error rates, queuing delays, throughput, etc.

In order to facilitate the segmentation and grouping of the user's data stream into data packets, the user may indicate in a full data packet whether there is a logical continuation of his data in the next data packet on a particular virtual circuit. This he does with the More Data bit (M) indicated in Figure 3. Only a full data packet requires a More Data indication since a partially full packet is treated as if it had the M bit off. The use of the M bit ensures that two communicating DTEs can each operate at their locally selected packet sizes.

Two independent mechanisms are provided to transfer control information between a pair of DTEs outside the normal flow of data. The first mechanism transfers control data within the normal flow control and sequencing procedures on a virtual circuit. This is called the data qualifier procedure. The format used in this procedure is identical to the normal data transfer packet except that the "Q" bit is set (see Figure 3). The data transmitted is then interpreted by the receiving DTE. An example of the use of the data qualifier is to transfer device control information such as echoing or packet forwarding rules and transmission control parameters for device-dependent interfaces on the packet network.

The second mechanism bypasses the normal data packet transmission sequence providing non-sequenced interrupt packets. Interrupt packets consist of a short header identifying the logical channel number and a one octet data field. Interrupt packets will be transmitted by the network without waiting for all other packets to be delivered and will be delivered to a DTE even when it is not accepting data packets. They contain neither send nor receive sequence numbers. In this way, interrupt conditions, such as would be generated by the depression of a break key on a keyboard terminal, can be signalled between DTEs without being subject to the flow control imposed on data packets.

Error recovery

The reset procedure is used to reinitialize the flow control procedure on a given virtual circuit to the state it was in when the virtual circuit was established (i.e., all sequence numbers equal to zero and no data in transit). To reach this state, all data and interrupt packets which may be in transit at the time of resetting are discarded. Reset packets are used in the reset procedure.

The restart procedure is primarily used by the DTE and provides a mechanism to recover from major failures. The issuance of a restart request packet is equivalent to sending a clear request on all switched virtual circuits and a reset request on all permanent virtual circuits. Thus, the restarting procedure will bring the user/network interface to the state it was in when service was initiated.

CONCLUSIONS

This paper has presented a description of a new communications protocol for accessing packet switching networks. The International Packet-Mode Interface has been developed by a number of administrations and common carriers in cooperation with standards organizations, users and manufacturers. We strongly believe that it is to the advantage of teleprocessing users, manufacturers of data processing equipment and common carriers that standards and recommendations continue in this area. The benefits to accrue are simplified design and use of equipment, lower cost, higher transmission efficiency, interconnectivity and enhanced performance.

The International Packet-Mode Interface has already been implemented in a number of installations to give access to the Telenet, Datapac and experimental RCP networks. It has been specified for use on the French PTT's Transpac network and for the Euronet international network being developed by a large number of European administrations.

REFERENCES

1. Chu, W. W., *Advances in Computer Communications*, Artech House Inc., Dedham, Massachusetts, 1974.
2. Cotton, I. W. and J. W. Benoit, "Prospects for the Standardization of Packet-Switched Networks," *Fourth Data Communications Symposium*, Quebec City, Canada, October 7-9, 1975.
3. CCITT Study Group VII Contribution No. 262, December 1975. Submitted by the French PTT and U.K. Post Office.

Virtual circuits vs. datagrams—Technical and political problems

by LOUIS POUZIN
Institut de Recherche d'Informatique et d'Automatique
(IRIA)
Rocquencourt, France

ABSTRACT

Public packet networks are becoming a reality, and call for interface standards. Two levels of facilities have been proposed, virtual circuit (VC), and datagram (DG). The concepts of VC and DG are already well developed within computer networks. Their properties are reviewed, along with typical issues such as out-of-sequence and congestion problems.

Usually DG's are a sub-layer used as a transport facility by a VC protocol. They also provide the ability to extend switching functions within user systems. The characteristics of VC's considered by CCITT are examined critically, and related to experimental networks and manufacturer softwares.

VC's and DG's are compared from the viewpoint of adapting customer systems to public networks. When the customer is interested in a transport facility, DG's appear to have an edge. When a network becomes a terminal handler, adaptations are more complex and require character stream interfaces. Intelligent terminals would make this problem disappear, as they can use a DG interface.

Although various groups call for a DG interface, the carriers are opposed to it. Four carriers are rushing a VC protocol through CCITT. The carrier's goal is to take over terminal handling, and gradually other processing functions. DG's would leave too much freedom to the customer. The political implications of the carrier policy suggest that better boundaries be drawn up between carriers and data processing.

THE COMMUNICATION INTERFACE

The development of large scale DP networks has pointed out the need for a well defined interface with a communication sub-system, which is termed here *a transport facility*. The term *transport* intends to emphasize the fact that this facility should move pieces of information from one place to another, without any alteration. Ideally, it should be error-free and carry data instantaneously. In practice, it introduces some transit delay and some errors.

In conventional systems, a widely used transport facility is a telephone line equipped with modems. The interface is standardized by CCITT (Comité Consultatif International Télégraphique et Téléphonique), and ISO (International Standard Organization). It is bit serial, at certain predefined speeds. New transport facilities such as *packet switching* are now becoming available. They are based on a combination of telephone lines and computers. The interface is more complex than for a simple modem. Typically it can be broken down into 3 components (Figure 1):

a—A modem interface, as usual;

b—A data link control procedure, such as HDLC[1] in charge of the transfer of packets between DP equipment and communication network equipment;

c—A packet protocol, in charge of sending and receiving packets according to a specific set of rules.

Components *a* and *b* do not raise any controversy. The modem interface is already widely available. The link control procedure is in the process of becoming an ISO standard. But component *c* turned out to be a tug of war.

Two approaches have been proposed for component *c*. One is a *virtual circuit protocol*, the other is a *datagram protocol.*

It is essential to note that the definition of a transport interface does not make any assumption on the

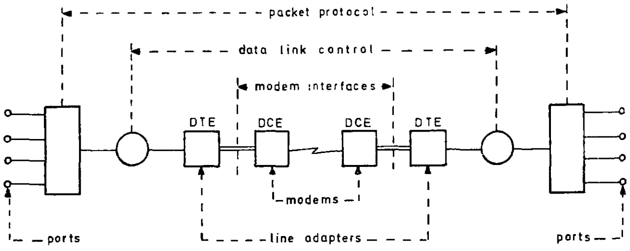


Figure 1—Packet network interface

specific techniques used to carry packets within the transport facility.

Even though some techniques may be more appropriate, with a certain technology, nothing prevents us from adopting widely different solutions, when new technologies become available. Changes need not, and must not, affect the transport interface.

VIRTUAL CIRCUITS IN GENERAL

A virtual circuit (VC) is a *logical path* between two end-entities for the purpose of exchanging data.

The terminology used in various groups is not stabilized, as is shown in Table I. In the following, end-entities are called *ports.*

Any computer network architecture embeds some form of VC, which is required to implement a *logical association* between a pair of correspondents. This is not so apparent when the pair of correspondents and their communication medium are represented by a dedicated set of resources, e.g., terminal control block, line adapter, telephone line, terminal. But it has become standard practice to share resources, such as a telephone line, between several pairs of correspondents. Thus, some machinery is required to segregate and control separately the traffic pertaining to each pair. This concept of *independent logical path,* along with an appropriate machinery, is what is called a VC, or many other things (Table I).

There has been a number of variations in the design of VC's. Since they are reviewed and discussed in a previous paper,[2] they are only given a short treatment here.

TABLE I—Virtual Circuit Characteristics

	COMPUTER NETWORKS				PACKET NETWORKS			
	ARPA Host net	CYCLADES EIN	IBM SNA	DECNET	EPSS PTT UK	TRANSPAC PTT FRANCE	TELENET USA	CCITT
VC name	connection	liaison	session	logical link	virtual call	circuit virtuel	virtual circuit	virtual call/ circuit
End-entity	socket	port	logical unit	object	customer label	voie logique	logical channel	logical channel
Sharable entity	no	yes	no	yes	no	no	no	no
Uni- or bi-directional	uni	bi	bi	bi	bi	bi	bi	bi
Error control	no	end	end	end	end	step	step	step
Flow control	end	end	end	end	step	step	step	step
Interrupt	yes	yes	yes	yes	no	no	yes	yes
Call Collision resolved	yes	yes	unknown	yes	no	no	unknown	no
Use DG's	no	yes	unknown	yes	yes	no	yes	not specified

Transit delay

Although delay presents no advantage, it is a built-in characteristic which cannot be eliminated. This is due to intrinsic response times and queueing always associated with resource sharing.

Uni- or bi-directional

It is the ability to transfer information in a single direction or in both simultaneously.

Permanent or transient

VC's may be set up permanently, or they may be opened and closed dynamically.

Sharable or busy ports

VC's are attached to ports bearing various names. In some systems several VC's may be anchored on a single port. In others, the port is busy when a single VC has been set up.

Call collision

In some systems, when two ports are attempting simultaneously to set up a VC with each other, one or both fail or two VC's are set up. This is a call collision. In other systems, which resolve collisions, this action results in a single VC being set up between the two ports.

Sequencing

VC's deliver information in the sequence it is sent. This may require a resequencing of fragments, when an underlying mechanism does not guarantee a sequenced delivery, e.g., a datagram network.

Error control

Most VC's perform error detection and recovery by retransmission. But there are two basic techniques:

—*step-wise,* in which control is applied onto successive legs of the physical path between ports (Figure 2). The reliability of the VC depends on intermediate mechanisms.

—*end-to-end,* in which control is applied by mechanisms located only at the two ends of the VC. Reliability is then independent of any intermediate mechanism.

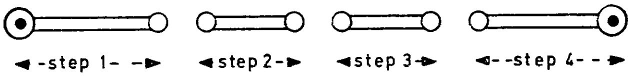


Figure 2—Step-wise error control

Both techniques may be applied simultaneously. Practically, end-to-end control always embeds some intermediate controlled legs. But there are step-controlled VC's without end-to-end control (Table I).

Flow control

It is the machinery in charge of keeping traffic within limits acceptable by the receiver, or any intermediate bottleneck. As for error control, two basic techniques are used:

—*step-wise,* in which feedback is applied on intermediate queues assigned to each VC;

—*end-to-end,* in which feedback is applied directly from the receiver to the sender. In this case, there is no need for handling intermediate queues per VC.

Again, both techniques may be applied simultaneously. However, when end-to-end flow control is used, intermediate legs are usually controlled in a more global manner (e.g., node-to-node, or network-to-network) rather than at the VC level. This saves a considerable overhead, and allows for *adaptive routing.*

Interrupt channel

It is used to by-pass the normal flow of information, when some signal must be transmitted even though error and flow controls may block up the VC.

Transport protocol

Sending and receiving information through VC's must follow a set of rules called a protocol. Basically, this protocol performs four types of functions:

(a) Implement a port name space in order to designate unambiguously sources and destinations at network level;
(b) Provide for the sending and the sequenced delivery of data, with error and flow control;
(c) If necessary, fragment items to be sent into

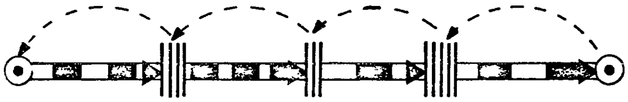


Figure 3—Step-wise flow control

pieces the size of the data field of a packet, and reassemble them at the receiver end;

(d) Multiplex the traffic of several VC's onto some common resources, e.g., a physical data link.

As may be inferred from the above, the sequencing functions require that all packets be routed through a minimum of two *focal points,* which insure numbering and sequenced delivery. Alternate routes can only appear in between the focal points, (Figure 4). The paths between ports and VC focal points must be unique and sequential. This creates a *reliability problem* when ports and focal points are not collocated in the same equipment.

DATAGRAMS

A datagram (DG) is a packet of information which is carried to its destination without reference to any other packet, or prior setting of a data path. In other words, a DG is a *self-contained* packet, in terms of switching.

As compared to VC's DG characteristics are considerably less diversified. The transit delay is of the same order of magnitude, or even shorter, since DG's do not have to wait for predecessors. All VC features, which are related to the concept of a logical path, are not applicable to DG's.

DG's can be sent to any port at any time. They are delivered in the sequence of arrival in the final queue to the destination port.

Since they are not related, they may be sent onto, and received from several physical data links in parallel.

The *DG protocol* consists only in formatting packets to be sent and dispatching received packets to specified ports. It may also include some conventions for flow control, when there exists a possibility of flooding some limited resource along the way.

Thus, DG's appear as a *very simple transport facility,* without mechanisms normally associated with an orderly and reliable transfer of information. An immediate question is whether DG's are useful.

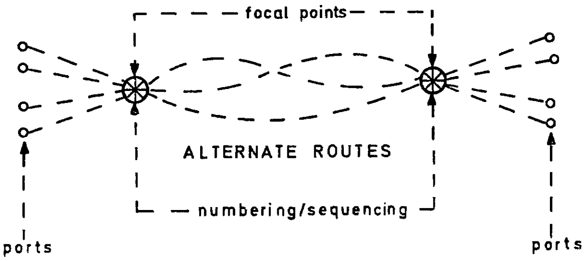


Figure 4—Focal points

At this point, it is interesting to quote a CCITT document[3] giving a definition for DG:

> "A user facility in which a message which can be contained in the data field of only one packet is delivered to the destination identified in its address field. No reference is noted by the network to any other datagram previously sent or likely to follow between the same two DTE's."

As long as the term *message* means everything or nothing, the definition is not incorrect, albeit meaningless. But if *message* is to have any correlation with a piece of data fit for processing, the definition is basically erroneous.

Indeed, neither VC's nor DG's can carry within a single packet messages longer than the maximum length of the data field (this is a tautology). Therefore, oversized messages are fragmented into pieces the size of a data field and sent as separate DG's.

At the destination, DG's are reassembled into a copy of the original message. Duplicates, if any, are discarded; missing DG's will be retransmitted if acknowledgment conventions have been established with the sender.

All this sounds very familiar, when compared to a VC protocol. But it is not a DG protocol. It is a user-oriented protocol interfacing the transport facility through a DG protocol. In other words, *DG's are not intended to be used as a self-contained transport facility.* On the contrary, they should normally be accessed through *an embedding higher level protocol.*

This higher level protocol may be a VC protocol, or *any other protocol* well suited to a specific class of applications. E.g., if user messages always fit within a DG, there is no need to invoke a fragmentation/reassembly machinery. This is likely to be the case of large scale applications, such as point-of-sale terminals.

The out-of-sequence issue

It is sometimes argued that DG's involve a substantial overhead at the receiver end to sort out packets, and put them back in the proper sequence. This is not the opinion of persons who have implemented protocols based on DG's. Indeed, in a sensible VC protocol using DG's, messages are broken down into labeled DG's carrying a message number and a DG number. All DG's of a message have the maximum length except the last one. On arrival, they are placed directly at their proper location in the message buffer, according to their number. This is utterly trivial, and needs no more comment.

It should be recalled also that resequencing is not peculiar to DG's. A sequential procedure, such as HDLC, delivers packets out of sequence when the command SREJ (selective reject) is used to trigger a retransmission.

Out-of-sequence packets could require more buffers than normally necessary if sequence shifts were frequent and had a large value. This is not the case in well designed networks. The reader may be interested in some simulation work[4] performed on this subject.

It certainly cannot be construed that out-of-sequence delivery carries an advantage per se. It is rather a non-issue. But the ability to accept out-of-sequence packets yields a major benefit in making *multiple physical links* a casual matter. This is indeed an essential ingredient if reliability is to be taken seriously.

The congestion issue

It has been recognized very early[5] that an excess of packets could use up all available buffers and result in deadlocks, or complete network jam, like cars in busy cities. Obviously, some mechanisms are necessary to prevent an overload of traffic. But the problem is not as simplistic as just keeping packets at bay. It breaks down into a number of sub-problems:

(a) Insure that packets do not enter the network when they would increase a developing congestion;
(b) Criteria to select acceptable packets when some of them must be held outside;
(c) What to do with packets that the receiver does not accept;
(d) What to do with packets that cannot reach their destination for any reason;
(e) How to anticipate congestion;
(f) How to optimize (?) network resources.

Most of these problems could apply to any resource sharing system, and it is well known that there is no general solution. But there are numerous partial solutions in a given environment.

It is often argued that a VC interface allows better traffic control. Actually, it only applies to *a*. No more.

Other schemes applicable to DG's have been studied and appear to be more effective, since they cope better with global congestion. The reader is referred to a very significant work accomplished at NPL.[6]

Every network designer cooks up some scheme coping with *a*, possibly at the cost of lowering the rate of resource sharing. E.g., in Transpac (French PTT), VC resources are dedicated all throughout the network.

Problem *b* requires some arbitrary policy including possibly tariff and marketing considerations. It has to be acceptable by the customer base. No general rule applies.

The most reasonable way to handle problems *c* and *d* is probably to destroy packets. As long as the destruction rate is lower than 10^{-3} or 10^{-4}, efficiency is not affected. End-to-end protocols handle easily DG loss. With a VC interface, the problem may require more extensive machinery, because VC's are not supposed to miss any packet.

Problems *e* and *f* are still a research area. They are independent of the network interface, be it VC or DG.

To sum up, there are a number of recipes to keep congestion under control regardless of which type of interface is used. Actually, carriers use DG's internally, which suggests that they are not so concerned. But making congestion control efficient is still an open question, for every network.

THE RELATIONSHIP BETWEEN VIRTUAL CIRCUIT AND DATAGRAM

As has been pointed out earlier, the DG is basically a low level transport facility at the disposal of higher level transport protocols, (Figure 5). It would be simplistic to oppose DG's and VC's, since they actually are complementary components in a hierarchical structure.

Indeed, computer networks such as CYCLADES[7,8] or EIN[9,10,25] use DG's as an underlayer of a VC protocol. This approach is in line with sound principles in system architecture, as it decouples the VC protocol from the characteristics of the transport facility. Thus, the VC protocol may operate as well over a thin wire, or a packet network, as long as they both transport packets.

Even the carriers have been able to understand the advantages of decoupling the VC protocol from the transport function proper. DATAPAC (Bell-Canada), EPSS (UK), TELENET (USA) are typical examples of packet networks based on this approach (Figure 6). Thus, it is all the more intriguing that carriers deny users the privilege to adopt the same principles.

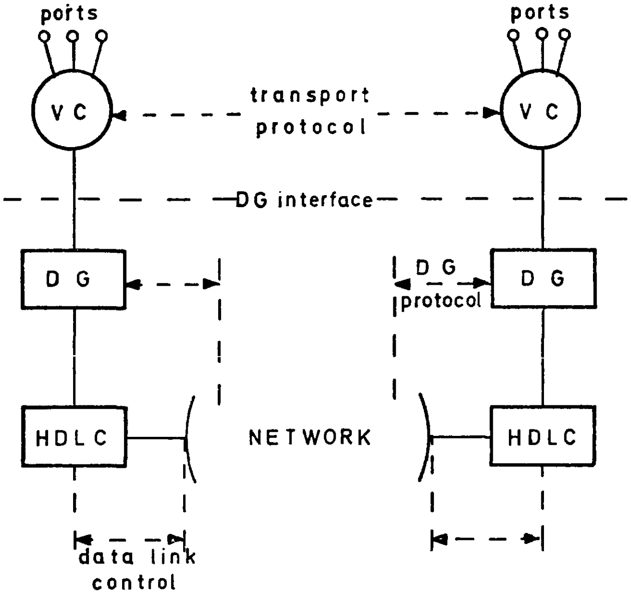


Figure 5—Protocol structure

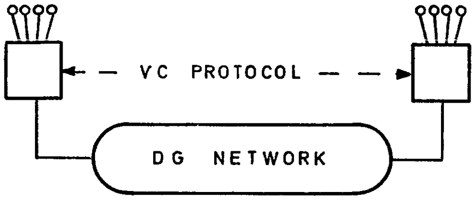


Figure 6—Two-level packet network

THE SWITCHING POTENTIAL OF DATAGRAMS

Another reason why carriers use DG's within their own packet networks is that switching DG's is much more efficient than handling logical paths all throughout a communication network. Intermediate nodes do not have to keep track of VC status; they only switch self-contained packets towards their destination. Error recovery and adaptive routing are much simplified.

Not surprisingly, the switching potential of DG's has also been put to an advantage by computer system designers. This stems from two simple remarks made earlier in this paper:

(a) DG's can be sent to, and received from several physical routes;
(b) DG's can be used as a transport sub-layer by several higher level protocols.

Therefore, the DG protocol acts as a *switch,* and interface converter, between user oriented protocols and transport facilities. It can also be used as an internal switch within a DP system. Nothing prevents us from putting several DG protocols, to deal with several transport facilities, public or private, or as internal relays within a distributed DP system, (Figure 7). This meshed structure provides for high flexibility and reliability, as it allows an arbitrary distribution of intelligence, without depending on a unique vulnerable component.

It can be verified easily that several recent computer system architectures include internal switching functions, carefully insulated from other parts of the system. They carry basic information units which are functionally equivalent to DG's. Interfacing with a DG network should require a minimum of adaptations.

Another benefit of the DG switching potential, is to simplify the interconnection of packet networks at DG level.[26] This approach has been extensively tested.[15] It allows multiple physical routes between networks. Interconnection at VC level requires additional focal points, which may render VC's inoperable in case of malfunctions.

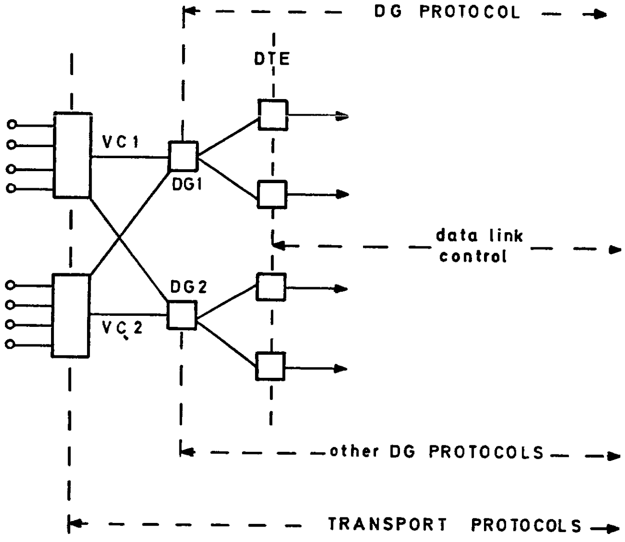


Figure 7—Multi-protocol Die

VIRTUAL CIRCUITS PROVIDED BY CARRIERS

During 1975, four carriers (France, UK, Bell Canada, and Telenet) evolved gradually a common proposal for a VC interface to public packet networks. Notwithstanding possible modifications, this proposal could become a CCITT *recommendation,* i.e., a standard, if it is agreed by a majority of carriers at the August 1976 plenary assembly.

At the present time, these four carriers have built, or are building, national packet networks with different interfaces. How and when they intend to migrate to the CCITT interface, is a moot question. Nevertheless, it seems more useful to concentrate on the CCITT draft,[11] instead of its ancestors. For ease of reference, we shall term CVC, (Carrier VC) the particular type of VC proposed to CCITT.

A subscriber equipment is called a DTE (Data Terminal Equipment) in CCITT terminology. A CVC may be summarized by its main characteristics:

(1) A DTE takes a subscription with the carrier for a certain number of bidirectional *logical channels* (LC) labeled 1 to N.
(2) In order to set up a CVC with another DTE, a calling DTE selects a local free LC number, and sends a calling packet to the carrier.
(3) Somehow, the calling packet reaches its destination within the carrier network (or another carrier network) to which the called DTE is physically linked.
(4) The carrier selects a supposedly free LC number and passes the calling packet to the called DTE.
(5) If the called DTE accepts the CVC, it returns an acknowledgment.

(6) The carrier propagates the acknowledgment back to the calling DTE.

(7) Both calling and called LC's are engaged for the time the CVC is set up.

(8) If two DTE's call each other simultaneously, two CVC's are set up, unless one of them (which one ?) rejects the call (unresolved call collision).

(9) Either DTE may decide to break the CVC at any moment.

(10) The physical link between DTE and carrier is controlled by an HDLC procedure (ISO standard). It must be full-duplex.

(11) The LC between DTE and carrier is controlled by a specific procedure that apes HDLC, without being identical.

(12) Packets sent by the DTE on an LC must not exceed a certain length agreed with the carrier.

(13) Packets delivered to the DTE on an LC do not exceed a certain length previously agreed with the carrier.

(14) Unless the maximum packet length used by both DTE's happens to be the same, there is no relationship between packets sent and received, except that they make up a continuous bit stream when concatenated by the receiving DTE.

(15) Message markers may be inserted by the sending DTE. This prevents the delivery of concatenated messages to the receiving DTE.

(16) Some CVC's may be set up permanently, subject to an agreement with the carrier.

It would be somewhat tedious to take into account all the various options anticipated. They do not alter substantially the general description presented here.

A few simple analogies may help to understand CVC's.

A DTE is like a PBX with one telephone number and a group of lines (LC's) to the public telephone exchange. A CVC is like a pipe, or a FIFO queue.

The CVC protocol provides only for interactions between DTE and carrier, not between DTE's. It is a *step-wise* protocol as far as error and flow control are concerned. This has strong implications:

(a) There is no way for a receiver to stop a sender, except by letting it fill up the CVC pipe;

(b) There is no way for a receiver to check that it has received its data completely, without alteration or duplication.

In other words: Thou shalt trust the carriers.

It is not clear why the CVC protocol comes on top of an HDLC procedure. They are both local protocols, controlling the transfer of data between the DTE and its nearest carrier office. If HDLC is presumed to work correctly, what is the purpose of a CVC protocol, and vice versa.

It may be argued that the CVC protocol needs more functions than HDLC. This is correct. But it would be perfectly adequate to control each LC with an HDLC procedure augmented with appropriate functions, for CVC set up, reset, and flow control. This is the way IBM does it in SNA. This would do away with one redundant layer, and bring the CVC protocol closer to an existing ISO standard.

However, due to the lack of DTE-to-DTE control, the *CVC protocol does not meet the qualification of a self-contained transport facility.*

As a consequence, users of CVC's will superimpose their own end-to-end protocol, such as a homemade VC protocol. Actually, this would happen anyway, because computer manufacturers and their customers must insure the reliability of their systems regardless of the vagaries of carrier facilities.

DATAGRAMS PROVIDED BY NON-CARRIERS

The concept of DG has been first experimented in the CYCLADES computer network, which uses a VC protocol[8] based on DG's. The CIGALE packet network[12] used in CYCLADES is a pure DG facility.

CYCLADES has been operational since early 1974. As opposed to Arpanet, the major part of the packet traffic is generated by remote batch jobs. At present, the bottleneck is the line printer, not CIGALE. This is an indication that DG's are an adequate transport facility in a remote batch environment. Conversational traffic is actually much less demanding.

The VC protocol has been implemented in various environments:

—in a user partition of a host computer,
—in a virtual machine under CP/CMS,
—within HASP,
—integrated with a telecommunication access method in manufacturer software,
—in a mini-computer as a host front-end,
—in a terminal concentrator,
—in a micro-programmed display controller.[13]

Another computer network, EIN, is being built on the same principles as CYCLADES. So far the participants have not identified any need for VC's at the packet network level.

An end-to-end VC protocol[14] based on DG's has been proposed by a working group of IFIP (WG 6.1). To date it is by far the best design which results from the largest exchange of ideas, and has received the widest acceptance. An early version[15] of this protocol has been in experimental use since mid 1974, between France and UK. Although the work accomplished in experimental networks has been generously disseminated, it has received little attention from the computer manufacturers, except for CII (Compagnie Internationale pour l'Informatique) in France, which has included VC's with a DG interface in its commercial software.

In addition, a packet switching software (i.e., DG) is available on a CII mini-computer.

Another computer manufacturer product, DECNET,[16,17] uses an end-to-end VC protocol independent of transport facilities, i.e., it can work with DG's.

IBM has released some documents on SNA,[18] in which there is an internal boundary with a transport facility. The transport interface appears to be of the DG type, but at the end of 1975, no information could be obtained on network structures allowing alternate paths. Furthermore, the transport functions are integrated in SNA. Facilities provided by carriers are limited to telephone lines.

VIRTUAL CIRCUITS VS. DATAGRAMS—TECHNICAL

As seen earlier, staging VC's against DG's in abstracto would be nonsensical, since they are independent layers. What is more relevant is to discuss the adequacy of either type of interface when provided by packet networks, namely public carriers. Thus, the discussion assumes CVC's.

It seems that all possible types of interfacing with public packet networks boil down to two classes:

—the carrier provides a transport facility in charge of transporting bits from one point to another;
—the carrier provides a device in charge of handling remote terminals.

This classification requires further elaboration.

Packet network as a transport facility

A typical situation is a private network in which dedicated lines and multiplexers are to be substituted with packet switching. What the customer is looking for is cheaper lines.

Only buffered equipment can send or receive packets. This rules out asynchronous character terminals, unless they are complemented with buffers. Normally, some kind of synchronous transmission procedure is used for the exchange of data.

First of all, delays introduced by a packet network may be unacceptable by application protocols. Assuming that this problem could be circumvented somehow, the customer would have to replace his transmission procedure with the one imposed by the network, i.e., HDLC, unless it is already there.

Then, the customer may subscribe for one or several CVC's for each piece of equipment, in keeping the same virtual topology. CVC's require the addition of the CVC protocol. If the customer is prepared to rely on the carrier for error and flow control, that's all he has to do. Otherwise, he would have to put yet another end-to-end protocol (E-E), which would give him complete control on the quality of the transmission on each CVC (Figure 8).

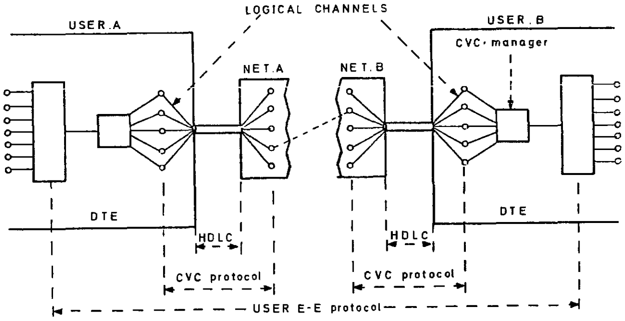


Figure 8—Virtual circuits as a transport facility

If the customer were using DG's, he would have to put an E-E protocol of his own choice, and a DG protocol.

To sum up, CVC's may have a minor advantage over DG's, when the customer agrees to trust the carrier, because a CVC protocol is perhaps a little more compact than an E-E plus a DG protocol. But this cannot be held true until it has been validated from experience for a number of protocols.

On the other hand, if the customer wants to insure his own E-E control, a DG interface would unquestionably take less overhead than CVC's.

As we have seen earlier, this latter situation is to become the rule rather than the exception, because computer manufacturers are introducing systematically E-E protocols in their network architectures. This approach has also been constantly advocated by users.[19,20,21,22,23,24] Unlike CVC's, E-E protocols within customer equipment are self-contained mechanisms. Furthermore, DG's provide for switching capabilities and multiple routes, which is a net improvement over CVC's in terms of flexibility and reliability.

Packet network as a terminal handler

This is the case where the customer wants to use a packet network to access non-intelligent character terminals. At one end he may have enough intelligence to implement whatever protocol would be required, but character terminals cannot transmit packets.

To that effect, carriers are providing functions to assist character terminals which they initially called packet assembling-disassembling (PAD). But a character terminal needs much more assistance than just PAD. E.g.,:

—character translation,
—recognition of break characters,
—timing of mechanical functions,
—translation of interrupt or error signals,
—recovery procedure,
—etc. . . .

It is no longer possible to control these functions over the packet network, since they are time dependent, and require a direct physical interface with the terminal. Therefore, the carrier has to do it. But terminals have a number of options, which the customer may want to use, or to vary. This requires some conventions whereby the customer, at either end, may set up his own menu. It is also nice to use different makes of terminals without having to develop special adaptations to the customer systems. Thus, terminals may access various computers. But a connect protocol and command language is necessary to specify the computer. And so on.

Step by step, the carrier is tempted to offer more and more facilities, which bear no relation to data transport whatsoever. These facilities are a set of functions which are logically at the level of an access method, such as VTAM. In a sound architecture, they should be matched by equivalent layers in the customer computer system. But, they are not. Flexibility has not yet reached that level.

Unless it is substantially modified, the customer system cannot communicate with the carrier as if it were dealing with a friendly terminal concentrator, because their access methods do not match. A typical scheme around this problem is to communicate with the carrier in terms of *character streams*, as if the carrier network were a terminal of a certain type, or a cluster of terminals.

Although this kind of interface is somewhat customized, some carriers appear to be willing to put the appropriate adaptation into their own networks, probably as the only way to get customers. Another variant is the customer developing a *gateway* between his main frame and the carrier.

With this approach, the customer delegates complete control of his terminals to the carrier. He has no way to check that things are running correctly save for hearsay from terminal users. This may be acceptable for certain classes of applications.

When the carrier provides the adaptation, the only interface visible to the customer is a character stream for a certain kind of terminal. Then, the question of using DG's or CVC's becomes an internal matter for the carrier. As we know, carriers use DG's internally to implement CVC's.

When the customer provides the gateway, he has to implement some protocol to exchange packets with the carrier. A simple DG interface would not be sufficient, as DG's assume an E-E protocol. Since the customer has no control on the other end, he has no choice but to use CVC's. Perhaps he would just wonder why the CVC protocol is that bungled.

But the real issue is not there. It is in the gateway itself. The hardware cost may be insignificant with large host computers. It certainly is significant when the host is a specialized mini. Furthermore, the largest part of the software is devoted to conversion chores, not to the CVC protocol. This means an inherent limitation in traffic throughput and a unique physical path to the carrier. Inter-computer communications become rather restricted if they have to mimic character terminal traffic.

A way out of that quagmire is intelligent terminals. If it turns out that they can be mass produced at a cost close to that of dumb terminals, there will be no need for terminal handling. Each terminal, or cluster, will contain enough intelligence to handle all necessary protocols, either locally or end-to-end. The most economic sub-layer to E-E protocols is a DG facility. As an illustration of that technique, a micro-programmable display[13] has been inserted in CYCLADES. It interfaces directly with the CIGALE packet network and contains all higher level protocols matching those in CYCLADES hosts. This puts terminal handling under complete user control, totally independent from the carrier. Thus, the packet network would revert to its initial destination: a transport facility.

VIRTUAL CIRCUITS VS. DATAGRAMS— POLITICAL

It should be apparent by now that there is something else than technical under the rug.

Indeed, manufacturers and users unanimously declare the view (as they say in CCITT) that there is a need for E-E protocols independent from transport facilities. This is an overwhelming position supported by scores of papers.[19,20,21,22,23,24,27,28] ISO has now started working on the standardization of an E-E protocol (TC 97/SC6/Project 17), which would allow data transfer between data processing equipments, *regardless of the transport facilities* used. IFIP-WG6.1 has already proposed such a protocol,[14] after extensive discussion and experimentation within network groups at international level. Therefore, one might think that the carriers would make every effort to take this approach into account in designing interfaces with public transport facilities, e.g., make it easy to superimpose E-E protocols on packet transport, and avoid the duplication of functions.

Some computer manufacturers have already expressed their preference for a specific transport facility: the DG. Others have not, and this includes IBM. Their position is that they cannot take a stand as long as specifications, quality of service, tariffs, are not known. Fair enough, because adaptation costs and effectiveness cannot be assessed without these essential parameters. But a certain number of institutions have already expressed a definite support for a DG facility. This includes IFIP-WG6.1,[27,28] the EIN community, the Industrial and Technological Directorate of the European Communities, the French Direction of Industry, and various users of large private networks.

Positions on CVC's are more polite. Everyone supports more or less CVC's without comment. It is not considered as a wise stand to antagonize the European PTT's when they are known to be ticklish (PTT's are telecommunications state monopolies). Manufacturers are even more cautious, since they always have some proposal hung up somewhere on a PTT desk. And after all, if CCITT standards are awkward, manufacturers will have a good excuse for selling their customers additional hardware. In case customers would then shy away from public packet networks, God bless the CCITT. Private networks are a bonanza.

Since carriers, and especially PTT's are primarily concerned with public interest, one would think that there are enough indications for providing both CVC and DG facilities. CVC's would be used by carriers as part of their terminal handling complex, and DG's would meet the requirements of a certain number of users. Actually, DG's are the only facility for which there has been real insistence coming from users and some manufacturers (not IBM). Software products using DG's are to be delivered in 1976.

A carrier's dream

As mentioned earlier, since mid 75, four carriers have been very busy coming to a consensus on a user interface with public packet networks (X25). (They are: French and UK PTT, Bell Canada and Telenet). They assume that CCITT, therefore the whole world, will buy up their common proposals at the next plenary assembly in August 1976. Due to administrative constraints, these proposals are practically firmed up at the beginning of 1976.

Conspicuously, the 4-tuple are adamant against DG's which they have downgraded from essential to optional, in opposition to the view of ECMA, IFIP, ISO,[30] and NTT.[29] Then, what is wrong with DG's?

Technical arguments are still used with non-experts, who may ignore that carriers use DG's for their own needs. The crux of the matter is elsewhere.

DG's call for E-E user protocols, which imply that the user is in control of both ends of the data path. In between, the carriers transport bits. But that is just a carrier business. Carriers do not want to be confined in carrier business any more. They want a more glamorous piece of the action: no less than a share of the computer market. According to their rationale, anything that goes on between a computer and a terminal is data transmission and switching. Since carriers have a monopoly in switching telephone circuits, they consider it a natural extension to demand the monopoly of virtual circuit switching for data. Terminals being a natural extension of circuits for testing, maintenance, and so on, should also belong to the carriers, like a phone set or a teletype.

Additional functions might be required, such as a command language, editing facilities, etc., but this is a natural extension to a dialing procedure. Carriers would standardize all that, and everyone would be happy. No less than a lion's share.

If the user were in control of both ends of the data path, nothing could ever happen, because he would keep buying total systems from his usual supplier. Therefore, it is essential for the carriers to be in control of at least one end of the data path. Clearly, working from the terminal up is a logical step. This is why such things as DG's, E-E protocols, are totally undesirable, unless they are completely under carrier control. In line with that approach, CVC's do not provide for E-E control at the user level.

An additional reason why DG's are outlawed is that they facilitate switching. That should be a carrier privilege. Therefore, it is essential that DG's do not spill away into the user domain. The CVC interface should be the Chinese Wall that contains DG's within carrier boundaries.

Once the carriers have secured terminal control in their networks, they will work at stealing more and more functions from the processing industry, to make them available within public networks. E.g., text handling, data banks,[31] data collection, software packages, about everything that is presently offered by service bureaus through private nets. Wherever possible, private nets will be outlawed or deterred with exorbitant line tariffs. State monopolies can do that.

Terminal manufacturers will have to queue up for months or years to get their products supported by public networks. Even if they conform to PTT specifications, a stamp of approval will likely be necessary. Innovation will be decided by PTT's.

Small computer manufacturers will be brought to compliance with CCITT standards and there will be no exception, because everyone else could put it up as a precedent. However, CCITT standards will not be incompatible with IBM, for practical reasons.

Carriers are huge bureaucracies. They have split viewpoints. Infighting goes on over issues like digital circuits vs. packet switching. Most carriers at top level are uncertain about choices made and would react negatively at the prospect of building a data processing empire, especially in Europe, where some of them cannot even cope with telephone. So, the packet clan has to walk a thin line, and tends to operate by political coups rather than by open policy. Occasionally, one of them is candid in telling the whole story.[32] Unfortunately, this is an exception. This is one of the reasons why discussions about public networks are somewhat eerie. In addition to the traditional secretive attitude of the European PTT's, there is a touch of deliberate covering up. It may explain such ludicrous documents where handling character terminals is just a matter of putting characters into packets, and vice versa. This is a model understatement for another Babel tower in the terminal jungle.

CONCLUSION

The controversy DG vs. VC in public packet networks should be placed in its proper context.

First, it is a technical issue, where each side has arguments. It is hard to tell objectively what a balanced opinion should be, since there is no unbiased expert. This paper argues in favor of DG's, but the author does not pretend being unbiased. Even if no compromise could be found, the implications would be limited to some additional cost in hardware and software at the network interface. So much resources are already wasted in computing and communications that the end result may not be affected dramatically.

Second, the political significance of the controversy is much more fundamental, as it signals initial ambushes in a power struggle between carriers and the computer industry. Everyone knows that in the end, it means IBM vs. Telecommunications, through mercenaries. It may be tempting for some governments to let their carrier monopolize the data processing market, as a way to control IBM. What may happen, is that they fail in checking IBM, but succeed in destroying smaller industries. Another possible outcome is underdevelopment, as for the telephone. It looks as if we need some kind of peacemaker to draw up boundary lines before we all get in trouble.

APPENDIX

ABBREVIATIONS

CCITT	Comité Consultatif International Télégraphique et Téléphonique
CVC	Carrier Virtual Circuit
DCE	Data Communication Equipment
DG	Datagram
DTE	Data Terminal Equipment
ECMA	European Computer Manufacturer Association
E-E	End-to-End
HDLC	High Level Data Link Control
IFIP	International Federation of Information Processing
ISO	International Standard Organization
LC	Logical Channel
NPL	National Physical Laboratory, Teddington, UK
NTT	Nippon Telegraph Telephone
PAD	Packet Assembling-Disassembling
PTT	Post Telegraph Telephone Administration
VC	Virtual Circuit
VTAM	Virtual Telecommunications Access Method (IBM)

REFERENCES

1. ISO/TC 97/SC6—*High Level Data Link Control Procedures,* Proposed Draft International Standard on Elements of Procedures, Doc. DP 4335, October 1975, 51 p.
2. Pouzin, L., "Virtual Call Issues in Network Architectures," EUROCOMP, Brunel Univ. September 1975, pp. 603-618.
3. *CCITT—COM VII—No 237-E,* October 1975, 142 p., Minutes of Meeting of Rapporteur's Group on Packet Switching.
4. Maier, M., "Out of Sequence Problem in a Packet Switching Network: A Simulation Approach," EUROCOMP, Brunel Univ. September 1975, pp. 191-206.
5. Davies, D. W. "The Control of Congestion in Packet Switching Networks, *IEEE Transac. on Comm.,* June 1972, pp. 546-550.
6. Price, W. L., "Simulation Studies of an Isarithmically Controlled Store and Forward Data Communication Network, *IFIP Congress,* August 1974, pp. 151-154.
7. Pouzin, L., "Presentation and Major Design Aspects of the Cyclades Computer Network," *3rd Data Comm. Symp. IEEE,* November 1973, pp. 80-85.
8. Zimmerman, H., "The Cyclades End-to-End Protocol," *4th Data Comm. Symp.,* October 1975, pp. 7.21-7.26.
9. Barber, D. L. A., "The European Computer Network Project," *ICCC,* Washington DC, October 1972, pp. 192-200.
10. Barber, D. L. A., *Progress with the European Informatics Network,* ICCC, August 1974, 15 p.
11. *CCITT—COM VII—Proposal for Revised Draft Recommendation X25,* France and UK Post-Office, November 1975, 67 p.
12. Pouzin, L., "Cigale, the Packet Switching Machine of the Cyclades Computer Network," *IFIP Congress,* August 1974, pp. 155-159.
13. Naffah, N., *Présentation du système Tipac. Doc. Cyclades,* Ter 524, November 1975, 15 p. French.
14. IFIP WG6.1—*Proposal for an Internetwork End-to-End Protocol,* September 1975, INWG doc. 96, 29 p.
15. Gien, M., J. Laws, and R. Scantlebury, "Interconnection of Packet-Switched Networks: Theory and Practice," *EUROCOMP,* Brunel Univ. September 1975, pp. 241-260.
16. Digital Equipment Corp., *Digital Network Architecture: Network Services Protocol,* July 1975, 44 p.
17. Teichholtz, N. A., "Digital Network Architecture," *EUROCOMP,* Brunel Univ. September 1975, pp. 13-24.
18. IBM, *Systems Network Architecture—General Information,* GA 27-3102-0, January 1975, 50 p.
19. Davies, D. W. "Principles of Packet Switching," *1st European Workshop on Computer Networks,* Arles, April 1973, pp. 175-190, IRIA edit. 78150 Rocquencourt—France.
20. Davies, D. W. "Packet Switching, Message Switching, and Future Data Communication Networks," *IFIP Congress,* August 1974, pp. 147-150.
21. Mann, D. W., "The State of the Art in the Evolution of Private Data Communication Networks," *EUROCOMP,* Brunel Univ., September 1975, pp. 415-432.
22. Chandler, A. S., "Network Independent High Level Protocols," *EUROCOMP,* Brunel Univ., September 1975, pp. 583-601.
23. Pouzin, L., "Standards in Data Communications and Computer Networks, *4th Data Comm. Symp.* October 1975, pp. 2.8-2.12.
24. ISO/TC97/SC6, *Documents 1145, 1167, 1173, 1249, 1258,* contain items on the structure of network protocols.
25. Poncet, F., J. B. Tucker, "The Design of the Packet Switched Network for the EIN Project," *EUROCOMP,* Brunel Univ., September 1975, pp. 301-314.

26. Pouzin, L., "A Proposal for Interconnecting Packet Switching Networks," *EUROCOMP*, Brunel Univ. May 1974, pp. 1023-1036.
27. IFIP-WG6.1, *Data Communications Standards,* May 1975, INWG Doc. 84, 15 p.
28. IFIP-WG6.1, *Basic Message Format for Inter-network Communication*, May 1975, INWG doc. 83, 7 p.
29. ISO/TC97/SC6, *Japanese Comment on User Facilities.* Doc. 1199, October 1975, 7 p.
30. ISO/TC97/SC6, *Doc. 1205*, October 1975, 36 p. This document includes some comments on user facilities, and an earlier draft of a virtual circuit protocol.
31. Fedida, S., "Viewdata: an interactive information service for the general public," *EUROCOMP*, Brunel Univ. September 1975, pp. 261-282.
32. Horton, D. J., *Presentation of Datapac. European Symp. on Large Scale Computer Networks*, Darmstadt, October 1975, 21 p.

Network access techniques—A review*

by ROBERT ROSENTHAL
National Bureau of Standards
Washington, D. C.

ABSTRACT

The computer industry's ability to serve a diverse and expanding user community is evidenced by the rapid growth of computer network services. Computer service providers design and market their own offerings as they deem best, given their own market and their own set of resources. This has led to a proliferation of similar resources requiring different user access procedures. With emphasis on currently operating and planned systems that assist users in accessing available network services, this paper identifies the techniques used in network access devices. By examining these devices, the trend toward improving the interface between the user and the computer is brought more clearly into focus and up to date.

INTRODUCTION

Little over a decade ago, when people began to interact with computers in the routine performance of their jobs, few cared about the differences between similar service offerings—all were eager to learn and experiment with that new technology.

Today, with the advent and growth of computer networks, that modest size group of scientists, engineers, and researchers has grown to include professionals in all sciences—mathematical, physical, health, and social—as well as students, stock brokers, and reservation clerks from many fields. Just as the number of users has grown so has the number and diversity of computer services.

Trends in service growth, traceable through individual families of mainframes, operating systems, and service packages, lack direction and consistency from the user's point of view. The reason for this fragmented growth might be justified considering constraints imposed by telecommunications facilities, peculiarities imposed by mainframes and their operating systems, and the personal preferences of system developers. Unfortunately, many of these services are characterized by different implementations of the logically similar steps that users must take in order to accomplish productive work.

Today we have a situation in which more and more users are consuming more and more services while the services themselves perpetuate differing user procedures for access to the same logical services. A reasonable question to ask is "What can be done to help the user?" Standardization of access procedures is a possible solution. However, a procedure for access that makes one service more attractive than another through enhanced features should not be compromised by a premature effort to standardize such procedures—especially if the standard settles on "lowest common denominator" features. Encouraging competition among network service providers can be in the user's best interest if it leads to innovation in the amount and quality of service received and in the reduction of costs in providing the service. Rather than push for extensive user-oriented uniformity, it may be desirable to continue to permit and encourage such non-uniformity, but to compensate for it through network access assistance to the user.

ASSISTANCE

The concept of assisting users is not new to the computer industry. The notion of a compiler, for example, resulted from this kind of motivation. Unfortunately, compilers were soon accompanied by complex operating system control languages. Once again, assistance techniques in the form of a control language macro capability or a catalogued procedure capability were employed to help the user. With these techniques the inexperienced user easily performed complicated job steps and the experienced user worried less about detail in setting up job steps.

This type of assistance, extended beyond a single computer system and placed in the interactive computer network environment, lessens the wearisome burden faced by users contending with separate and

* This work was sponsored in part by NSF grant DCR 72-01206 A06.

different services that accomplish logically similar tasks. In such an environment, the user should have resources readily available from different computers without regard for the specifics of how to obtain them; the user should be more concerned with what services are required.

Early attempts to provide computer network assistance are still in use today. One interesting technique makes use of function buttons that produce character sequences. These sequences represent the appropriate user protocols that identify terminals, users, and services, thus alleviating the need for the user to key in the required protocol. The familiar "who-are-you" drum is an example of this kind of technique. Other devices—stunt boxes, automatic card dialers, and paper tape loops—have been used successfully to assist users, but all of these devices are usually applicable only on dedicated connections to specific computer systems and services. In a modern computer network other more general techniques can be employed.

INTERFACES AND PROTOCOLS

Access assistance techniques in modern computer networks try to improve upon the interface that exists between a user at his terminal and a network based resource. For the terminal user of an interactive computer network the interface is two things. First, it is the physical equipment—the teleprinter or CRT terminal and the communications equipment connecting it to the computer. Second, at a higher and more complex level, the interface is the protocol that a user must know to communicate with the network and its computers—to express his needs or demands on the computer and to understand the produced results or errors from the computer.

These user protocols, unlike the "well defined" link protocols of computer-to-computer communications (such as the ARPA Network IMP-IMP, HOST-IMP, or HOST-HOST protocols,[1]) manifest themselves in the interactive dialog between the user and the computer. And, these user-computer protocols are not "well defined." They are typically machine dependent and often installation dependent.[2]

Interface and protocol standards groups are actively engaged in producing outputs directly related to the issue addressed here. At least one standards group in the Federal Government (FIPS Task Group 20) is studying low level user entry and exit protocols and procedures. The standards approach to access assistance represents a technique that is still in its infancy. With a three year projected completion date for the development and acceptance of the first entry and exit protocol standard for the Federal Government, the viability of this technique is yet to be realized. The beneficiaries of standards as an assistance technique will be the scientists, engineers, researchers, and those who work with them, who use or have the potential for using one or more computer services.

TECHNIQUES

These approaches to solving the problem of network access have been utilized by several groups in specific implementations of the access function. Pyke, in a recent paper,[3] reviews these efforts with special emphasis on presently operating and planned access support configurations. Categorizing these examples and other related devices as reported in the open literature reveals the trend that is evident in network access techniques—a trend toward improving the interface between the user and the computer through sophisticated assistance techniques and devices. By examining these devices—their methodology, their purpose, and their scope—this trend is brought more clearly into focus and up to date.

Basic communications assistance

Minicomputer based concentrators and packet or message switchers are well established solutions that provide the basic communications required of network users. Devices like the ARPA Network TIP[4] and the TYMNET Network TYMSAT[5] satisfy these requirements through data rate identification, terminal compatibility transformations including carriage delay timing functions and character set transformations, and host computer selection specifications.

Over the last few years these devices have been modified to reflect the needs and demands of the user community. They utilize software packages that map character sets and provide carriage delays to accommodate different user terminals. These devices continue to evolve in an attempt to follow manufacturers' terminal innovations. In response to new terminal innovations, devices like the TIP and TYMSAT recognize the user's terminal speed; while default terminal characteristics for other parameters—parity, full or half duplex, tab settings, etc.—are assumed, provisions to specify different settings are available. Many computer service providers and mainframe manufacturers have followed this trend by providing communications support for a large variety of terminals and by utilizing front end computers with appropriate software packages.

Resource identification

Access techniques only start with the basic communications assistance functions described above. Establishing connections to network host systems, logging into host systems, requesting resources, and initializing services or databases can be extremely complex and cumbersome. In one early ARPA Network attempt, a loose leaf notebook containing information on the resources available in the Network and on the access methods to these resources has met with only very limited success. The Network Information Center's ARPANET Resource Notebook[6] existed in both an on-line and printed form and contained entries for all of the resources available at each serving host on the network. Unfortunately, no attempt was made to facilitate access to these resources other than listing individuals to contact. (A more recent edition of this notebook has been published in paperback form[7] and includes enough information for a user to access the desired resource.) Resource identification techniques of a more substantive nature developed.

The REX system, an ARPA Network based on-line user assistance facility, provides resource-specific information.[8] This system, designed with the eventual goal of automating access to various network resources, finds the location of a resource on the network, provides information about the resource on the network, and describes the method for acquiring the resource on the network. REX provides a facility for dealing with a heterogeneous network as a coherent entity, regardless of the particular characteristics of the individual hosts.

A user language that provides commands to retrieve information about resources and to describe specific resources utilizes one of four keyword types—a resource name, a resource attribute, a resource category, or a host name—in conjunction with either a retrieval command—FIND or DESCRIBE—or the ACQUIRE command. An example is:

```
DESCRIBE FORTRAN AT MIT-MULTICS
```

The ACQUIRE command establishes a transparent connection to the resource without further action on the part of the user.

Resource connection

Connections to resources that are available in computer networks require cooperation between the connection initiator—the user or his surrogate—and the resource that ultimately provides the service—a coherent logical entity that exists in order to accomplish a specific task as viewed by the user. In a computer network environment it is not unusual for a multi-level hierarchy of access requests to occur to affect resource connection. Cooperation in traversing the network hierarchy may require several resource solicited identifications, passwords, and system or service names.

While the solicitation of user information occurs at the user level of interaction, many other protocols are used by the communications discipline employed. Usually these other disciplines are masked from the user and are transparent. In this way, a surrogate—usually a process knowledgeable in the protocol or line discipline employed—acts on behalf of a user to connect him to the requested service. For instance, users of the ARPA Network access resources with the cooperation of a TIP. It is feasible for the acquired resource to then request other resources on behalf of the user. In this example, the user level of interaction is the command language of the TIP; acting as a surrogate for the user, the TIP, knowledgeable in the IMP-IMP protocols, completes the resource connection for the user.

Hierarchical connections easily occur. In the ARPA Network, many host computer resources support a service known as FTP (File Transfer Protocol). This service establishes network connections to other resources on behalf of the user for the purpose of transferring files between host computers. Currently, the user need not identify himself to the TIP, however a user account is required for the host system to execute the service FTP. For FTP to access the file system of the other resource, the user level protocols require the user identification and password for the other resource.

Resource selection

Evidence in the published literature indicates that a trend in the use of intelligent terminals and their close cousins," "clustered" terminals, to assist users in the selection of resources is developing. A research program currently under way at the RAND Corporation aims to develop a prototype intelligent terminal system initially implemented on a minicomputer. The system, called the RAND Intelligent Terminal Agent (RITA), is based on sets of condition-action rules that encode complex sets of heuristics for handling interactions both with users and with external systems.[9]

RITA is capable of interacting with remote data systems, carrying out time-dependent tasks over extended periods of time in a semi-autonomous manner. The supposition that a rich set of heuristics for deciding communications levels of remote system interaction and for resource acquisition and graceful recovery from unexpected failure has in part motivated this effort.[10]

At the National Bureau of Standards a minicomputer-based device called a Network Access Machine (NAM) expands user-entered commands into command sequences executable on specific networks and host computers connected to that network.[11] The NAM analyzes system and network responses to assure agreement with those anticipated for specific commands. Conditional and parameterized expansions of user-entered commands are capabilities being added to the basic working NAM. This capability will allow the use of the same commands to permit access to resources on different host computers and different networks.[12]

Other extensions to the NAM are currently being implemented. One function is to predict the expected response time of a particular computer for use in a specific applications area—such as compute-bound FORTRAN jobs, small BASIC jobs, interactive editing sessions, or some other category. Good indicators of computer response time can be calculated by the NAM in the following way. The NAM automatically connects to the particular computer in question in order to execute a predefined benchmark job representative of the applications area. By having the NAM automatically time the response for the job execution the NAM can present to the user the results of the calculations in the form of a prediction of expected response time. Several benchmarks have been successfully tried—producing excellent predictions of expected response time.

One other intriguing function being added to the NAM is to provide the user with network wide tutorial assistance. A profile, maintained for each NAM user, reflects the disposition of the user with respect to any particular computer that he might use. Data are maintained in the profile concerning recent use of specific computers, the use of internal indicators, and help assistance indicators. Using this data base, the NAM acts on behalf of the computer in assisting the user.

An example of somewhat specialized access support to a network user employing a microprocessor is the development of a "line processor" by the Stanford Research Institute. This device interfaces a class of display terminals together with a pointing device called a "mouse" and a one-hand keyboard called a "key set" in such a way that the entire configuration can provide particularly effective access to a network-based interactive text manipulation system. The line processor has potential, however, for application beyond the initial intended use.

Service integration

An important trend, motivated by the desire to interface one user to multiple resources, is evident in the work done in the information services community by Marcus at MIT. He has developed a system that involves the coupling of two bibliographic data retrieval systems in such a manner that the user perceives a single homogeneous system.[22] Specific applications support built within the MULTICS system at MIT includes a master index and thesaurus that stores the vocabulary of the separate data bases along with index term interrelationships. The user is also provided with a common bibliographic data structure in which the data elements for bibliographic information are organized and interrelated among different data bases. This approach has been demonstrated experimentally using the ARPA Network for access to the National Library of Medicine Medline service and the MIT Intrex retrieval system.

Another example is the Resource Sharing Executive (RSEXEC) for the ARPA Network.[13] This executive system provides an environment for "inter-host" user-user interactions, for managing "multi-host" file directories, and for controlling multiple "jobs" on several hosts. In addition, the RSEXEC serves as a command language interpreter for the ARPA Network TIP users. Executing in any one of several available PDP-10 TENEX systems on the Network, RSEXEC can maintain communications through the Network with other TENEX systems. In this way the resources of all of the systems to which RSEXEC connects can be monitored and used. Facilities for monitoring status, logged in users, and load averages are available as well as the capability to build and create a composite file directory incorporating files from the various computers in the subnetwork. Provision is also available for initiating jobs at one or more cooperating sites at which valid accounts are maintained.

Distributed assistance

Distributed assistance—partially in host systems and partially in front end systems—can provide better network response at less cost. The National Software Works (NSW) motivated by a desire to provide users with access to a number of general or specialized software applications packages, resides on a PDP-10, but will also be available on a minicomputer such as the PDP-11.[14] NSW provides assistance in such a way so as to maintain a consistent user interface from package to package even across host systems. It is anticipated that heavily used common commands and grammars can be executed in the front end systems, thus increasing network response times and decreasing network communications costs.

Kimbleton and Schneider emphasize that the NSW is intended to be an environment for building software systems.[15] It is to include program preparation tools—cross assemblers and compilers, editors, simulators and emulators, performance analyzers, program formatters, flowcharters, test-data generators and other checking tools—that build software systems. These systems will usually be removed and run in a different environment, since the NSW is not intended to support the recurring execution of end-user applications.

Minicomputer hosts

A minicomputer installed as a network host computer can also perform substantial access functions for a community of users. The ELF system provides multiple, concurrent users with local computing and file capability including signal processing for speech applications as well as flexible access to ARPA network resources.[16] The ARPA Network Terminal System (ANTS) was a mini-host designed to facilitate use of the ARPA network by students at the University of Illinois. Use of this system promoted RJE use of the Burroughs 6700 at the University of San Diego as well as RJE use at the Campus Computing Network (CCN) at UCLA.[17]

At the University of Chicago, Ashenhurst has developed an interesting hierarchy of minicomputers and large scale computers to assist users in laboratories.[18] Minicomputers located in science laboratories throughout the campus are served by a larger minicomputer at a central site. Large compute-bound applications are served when the central minicomputer acquires resources on a large scale computer on behalf of the laboratory mini.

Other trends

The work of Wyatt at Harvard, in planning for an access system to couple users to multiple serving systems and networks, includes a call for automatic transformation of job control statements to match those of remote systems and for facilities to support "transparent connection" of local terminal facilities to various communications networks.[19] Wyatt also proposes that the "translation/communication system" perform comprehensive accounting and billing for multiple user accounts.

ARCHITECTURE

The preceding examples suggest that alternative solutions exist and that the access assistance functions can reside in a process executing in a host system, in a dedicated minicomputer, or in a single user "intelligent" terminal or terminal "cluster."

When supported on a large host computer, the access assistance functions utilize the sophisticated and extensive subsystems that are available—the file management systems, the language compilers, the utility processors and on and on. But, this type of application can be accomplished with less sophisticated resources. For instance, the ARPA Network TIP, in part motivated by a desire to move the basic communications support and low level access functions out of the large host computers and into smaller, less expensive minicomputers, eliminates much of the overhead required to handle access functions in a larger host system. Minicomputers are often used extensively for store-and-forward functions and other applications such as remote concentrators and message switchers; it might make good economic sense to put the access function in a minicomputer also.

The minicomputer, placed in the communications link between the user terminal and the serving computer network, can allow multiple users to access different hosts on one network or even different networks. The minicomputer seems ideally suited for such an application especially when several users or user groups require a moderate file storage capability for small files and a minimum computational capability for communications processing, signal processing or other requirements.

Intelligent terminals can support access to multiple hosts on a single network or to multiple networks, but file storage and other terminal resources appear to be too expensive at this time to dedicate to a single user terminal. Also, intelligent terminals may require cooperation with larger host systems for initial program loads; such cooperation may not be available from candidate networks or host computers. At this time, the intelligent terminal does not have as great a potential as the "cluster" supported minicomputer to perform the required access functions. Based on predictions that within five to eight years inexpensive interactive terminals will be available with the power of today's minicomputers, the use of intelligent terminals to support the access function will be feasible.

SUMMARY

Increased use of computer services by people in the routine performance of their jobs continues to grow steadily. To meet the demand, service providers design, implement, and market their services using the technologies and resources available to them. With the advent and growth of computer network technology it is not uncommon for people to use the resources of different computer systems. The non-uniformity in access to these resources and the services provided has led to the development of tools and techniques to assist the user.

Some of the tools and techniques that provide users with this access function have been described in this paper. By examining these devices—their methodology, their purpose, and their scope—and by discussing the current trends in network assistance architectures, a framework has been built for future discussions of this very important function that can be provided to computer network users.

REFERENCES

1. Postel, J. B., et al., *ARPA Network Current Network Protocols*, Network Information Center NIC 7104, December 1974 (distributed by NTIS AD/A-003 890).
2. Neumann, A. J., *User Procedures Standardization for Network Access*, NBS Technical Note 799, October 1973.
3. Pyke, T. N., Jr., "Network Access Techniques: Some Recent Developments," *Proceedings Third Annual Texas Conference*, October 1974.
4. Ornstein, S. and M. F. Heart, et al., "The Terminal IMP

for the ARPA Computer Network," *Proceedings SJCC 1972*, pp. 243-254.

5. Tymes, L., "TYMNET—A Terminal Oriented Communications NETWORK," *Proceedings SJCC 1971*, pp. 211-216.
6. *ARPA Network Resources Notebook*, The Network Information Center NIC 6740, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, 1973.
7. *ARPANET Resource Handbook*, (NIC 23200), Network Information Center, Stanford Research Institute, Menlo Park, California, 1975.
8. Benoit, J. W. and E. Graf-Webster, "Evolution of Network User Services—The Network Resource Manager," *Computer Networks: Trends and Applications*, IEEE Inc., New York, 1974, pp. 21-24. (Proceedings of the 1974 Symposium sponsored by the National Bureau of Standards, Gaithersburg, Maryland and the IEEE Computer Society).
9. Anderson, R. H., "Advanced Intelligent Terminals as a User's Network Interface," *How to Make Computers Easier to Use*, (COMPCON 75 Digest of Papers, September 9-11, 1975) IEEE, New York, 1975, pp. 180-182 (IEEE Catalog No. 75ch0988-6c).
10. Anderson, R. H. and J. J. Gillogly, "The RAND Intelligent Terminal Agent (RITA) as a Network Access Aid," to be published in the *Proceedings of the National Computer Conference*, 1976.
11. Blanc, R. P., "Assisting Network Users with a Network Access Machine," *Proceedings ACM*, November 1974.
12. Rosenthal, R., "Accessing On-line Network Resources with a Network Access Machine," *Access to Computer Networks* (Session 25, IEEE Intercon 75, IEEE Intercon Conference Record) IEEE, New York, April 1975, pp. 25/3: 1-4.
13. Thomas, R. H., "A Resource Sharing Executive for the ARPANET," *Proceedings of 1973 National Computer Conference*, June 1973, pp. 155-163.
14. Irby, C. H., "The Control Meta Language System," to be published in the *Proceedings of the 1976 National Computer Conference.*
15. Kimbleton, S. R. and G. M. Schneider, Computer Communication Networks: Approaches, Objectives and Performance Considerations," *Computing Surveys*, Vol. 7, No. 3, September 1975, ACM.
16. Retz, D. L., "ELF—A System for Network Access," *Access to Computer Networks*, (Session 25, IEEE Intercon 75, IEEE Intercon Conference Record) IEEE, New York, April 1975, pp. 25/2: 1-5.
17. Bouknight, W. J., G. R. Grossman and D. M. Grothe, "The ARPA Network Terminal System—A New Approach to Network Access," *Proceedings DATACOM 73*, 1973, pp. 73-79.
18. Ashenhurst, R. L., "Hierarchical Computing," in M. Greenberger, et al., (eds.) *Networks for Research and Education*, the MIT Press, Cambridge, Massachusetts, 1974, pp. 74-88.
19. Wyatt, J. B., "Management in Applications of Network Access," *Access to Computer Networks* (Session 25, IEEE Intercon 75, IEEE Intercon Conference Record), IEEE New York, April 1975, pp. 25/1: 1-6.
20. Kriloff, H. Z., "A High-level Language for Use with Multicomputer Networks," *Proceedings of 1973 National Computer Conference*, pp. 149-153.
21. Luther, W., *Conceptual Bases of Cybernet, Computer Networks*, ed. by Rustin and Randal, Prentice-Hall, Inc., New Jersey, 1972, pp. 111-146.
22. Marcus, R. S., "Network Access for the Information Retrieval Application," *Access to Computer Networks* (Session 25, IEEE Intercon 75, IEEE Intercon Conference Record) IEEE, New York, April 1975, pp. 25/4: 1-7.
23. Pyke, T. N., Jr., "Some Technical Considerations for Improved Service to Computer Network Users," *Proceedings COMPCON 73*, February 1973, pp. 53-55.
24. Roberts, L. and B. Wessler, "Computer Network Development to Achieve Resource Sharing," *Proceedings SJCC 1970*, pp. 543-549.
25. Reintjes, J. F. and R. S. Marcus, *Research in the Coupling of Interactive Information Systems*, Final Report No. ESL-FR-556, Electronic Systems Laboratory, Department of Electrical Engineering, MIT, June 1974.
26. Rosenthal, R. and S. Watkins, "Automated Access to Network Resources: A Network Access Machine," *Computer Networks—Trends and Applications*, NBS-ICST and IEEE Computer Society, May 1974.
27. Balzer, R. M., T. E. Cheatham, S. D. Crocker and S. Warshall, *National Software Works Design*, USC/Information Sciences Institute, RR-73-16, November 1973.
28. Carlson, W. E. and S. D. Crocker, "The Impact of Networks on the Software Marketplace," *EASCON 74 Record*, IEEE Electronics and Aerospace Systems Convention, IEEE, New York, October 1974, pp. 304-308.

The RAND intelligent terminal agent (RITA) as a network access aid

by ROBERT H. ANDERSON and JAMES J. GILLOGLY
The RAND Corporation
Santa Monica, California

ABSTRACT

An operational "intelligent terminal agent" system called RITA is described. RITA uses production systems to store heuristics about dealing with the interactive protocols of external information systems. The use of production systems allows RITA to operate in either a pattern-directed or goal-directed manner. By creating new production rules as part of their operation, RITA agents can exhibit learning. Advantages and disadvantages of production systems for creating intelligent terminal agents are discussed, and an annotated transcript of a session with a RITA agent is given to illustrate its ability to aid a user in handling File Transfer Protocol on the ARPANET.

INTRODUCTION

Until recently, most users of information systems have been either "computer sophisticates"—such as computer programmers—or else users of systems, such as airline reservation systems, with a very limited set of options. However, with the continuing rapid decline in the cost of computer hardware and data communications, many interactive computer-based information systems are becoming cost effective for much broader categories of users. These newer systems will greatly expand the number of people interacting with computers in their daily activities, and will give access to a complex variety of interactive protocols, interfaces, command languages, and remote computing systems. People are going to need assistance in tailoring this variety of options to their specific needs and especially in freeing them from routine interactions and protocols which are not directly relevant to the content of their task.

We assume the complexities of dealing with the interactive protocols of computer networks are familiar to this audience, and are sufficiently documented[1] not to require further elaboration. This paper describes one possible solution to the problem of complexity: user access to computer networks aided by an "intelligent terminal agent."

What is an intelligent terminal agent and why is it useful? The answer involves many aspects of the way people interact with computer systems as well as new alternatives becoming possible through advances in both hardware and software technology. The following observations concerning man/machine interaction and its supporting technologies form the basis for our design of such an agent:

1. Projected computer hardware cost trends and advances in microprocessor technology make it extremely likely that interactive computer terminals can be produced within five to seven years containing processing power and data storage equivalent to a present-day minicomputer, at a cost which is reasonable, assuming fairly intensive use of dedicated terminals by professionals as part of their job.
2. It is important to have certain information storage and handling capabilities locally*—most likely within the terminal itself—e.g., to provide "instantaneous" response to simple text manipulation commands and to simple error conditions.

Once local computing power becomes available within a terminal, it can be used to aid in interfacing with external information systems, such as the ARPANET or New York Times Information Bank, where much of the interactive protocol involves supplying standard responses which are not directly relevant to the task being accomplished. It is possible in the system described in this paper to teach an intelligent terminal to deal with such interactive protocols automatically, including instructions on dealing with certain error conditions, so that these details need not be remembered and handled manually by the user.

In addition, intelligent terminals should allow the user to define and set in motion "user agents". Such an agent could:

> Look at a calendar of events and start up services for the user automatically at certain times and dates. By manipulating calendar items, the human manager can progressively modify the plan

* "Locally" is used here to mean computing power and storage dedicated to the individual user and accessible via a very-high-bandwidth link—e.g., sufficient to rewrite a 3,000-character CRT display in 0.5 seconds.

being executed by the machine. For example, by changing the due date of a report, the schedule will be automatically altered for reminders and follow-up queries to be transmitted to persons making contributions.

Monitor the occurrence of various types of events, such as the arrival of a certain piece of network "mail", or the occurrence of a certain datum in a changing data base.

Deliver "interactive letters" to other users' terminals; these letters are capable of carrying on a dialog with the recipient, while in the process extracting information from him in a standard format suitable for further automated processing. Reference 2 contains an example of an interactive letter.

Manage transactions between a number of computerized services distributed on a computer network, monitoring their successful accomplishment.

We believe the desirability of the above list of services is a compelling reason to explore the design of intelligent terminal agents capable of running in present-day minicomputers. The remainder of this paper describes the design and operation of a system—the Rand Intelligent Terminal Agent (RITA)—which has been developed at Rand to meet all of the above requirements. RITA is currently operational on a PDP 11/45 minicomputer running the UNIX operating system.[3] The next section discusses "production systems" as the software technology underlying RITA. The third section contains examples of RITA's operation as an interface to computer networks. The fourth section concludes with a discussion of RITA's implementation status and some remaining research questions which have been raised by our work to date.

PRODUCTION SYSTEMS AS THE BASIS FOR A TERMINAL AGENT

The basis for our design of a system meeting the above requirements is the use of production systems. A production system consists of a set of production rules (having a pattern part and an action part), which operate upon a data base (which we call a context), according to the actions of a rule interpreter, or monitor.

For example, two production rules might be:

Rule 1

IF: there is a message whose status is "awaiting action" and the identification-field of the message is not in the action-items of the user

THEN: put the identification-field of the message into the action-items of the user;

Rule 2

IF: the latest-command of the user is "show action items" and the state of the system is "command unfulfilled"

THEN: send the action-items of the user to the user and set the state of the system to "command fulfilled".

These rules would be part of a larger set of rules governing a message-handling user agent. They might be interpreted by a monitor that continually tests the "if" conditions in each rule of the set, and executes the "then" actions in any rule whose conditions are all true. Assuming messages with various attribute values, such as an identification field and status, are placed in the data base by some external (and possibly asynchronous) process, the above rules would update a list consisting of the identification numbers of all messages awaiting action, and show that set to the user upon his request. Other rules would themselves, or permit the user to, take other actions and as a consequence change the status of the messages and remove their identification number from the set of action items.

There are a number of interesting options in the design of production systems. For example, production rules can be used in either a goal-directed or pattern-directed manner. A goal-directed system has a designated goal, and the objective is to execute rules whose actions help to achieve that goal. These two modes of operation are discussed further under the heading "Monitors," below.

A good discussion of design options in production systems is contained in a recent survey article by Davis and King.[4] That reference, as well as Reference 2, can be consulted for more details. A series of articles on the MYCIN system[5-9] by E. H. Shortliffe and associates at Stanford University describes a particular goal-directed production system which has significantly influenced the design of the RITA system.

Our design decisions in creating a production system for our particular needs are discussed below under four headings: data base, rules, monitors, and system architecture.

Data base

The data base upon which RITA rules operate is called a context; it consists of an unordered set of objects. Each object has a name, or type, and there can be more than one object in the context of the same type. There is neither an external structure imposed on the set of objects in the context nor a requirement that each object have a unique identifier associated with it. Each object can have one or more named attributes, and all attributes attached to an object must have names which are mutually distinct. Each attribute has an associated value, which is either a character string or an ordered list of values.

Objects, attributes, and values may be created or deleted dynamically by the actions of rules. If an attribute being tested by a rule's predicate does not exist, it is considered to be "not known." It is possible by a rule action (except within a goal-oriented monitor) to reset an attribute having a value back to the "not known" status. Goal-oriented monitors may not reset the value of any attribute; they may only set values which were previously not known. This restriction is necessary to preserve the integrity of information upon which chains of logical reasoning are based.

As an option, it is possible to attach a "level of certainty" to a string attribute value as it is being set. In this case, an attribute can have several different values associated with it, each with a different level of certainty. Levels of certainty are adjusted as additional positive or negative certainty factors for those values are asserted by the action of rules. Our use of certainty factors has been strongly influenced by their implementation in the MYCIN system, but differs in some details which will not be discussed here.

Figure 1 contains examples of object types and associated attribute names and values which might be used in a user agent within the RITA system.

The data structure we have chosen is not the most general one possible. As with other implementation decisions, we have chosen what we consider to be the simplest format and conceptual structure which allows the description of situations and heuristics related to intelligent terminal agents. With more experience in using the RITA system, some of these decisions are almost certain to change.

Rules

RITA rules are expressed in a finite syntax (technically, parsable by an LR(1) algorithm). We have chosen a syntax patterned after the specialized English output form generated to display MYCIN rules to a user. We believe that this syntax is simple enough to be read and written by a computer-naive user. Figure 2 contains several examples of clauses which can be used in RITA rules; more complete examples are contained in the transcript in a later section.

object type	attribute name	sample value
file	name	"foo.baz"
	directory	"jjg"
	site_id	"rand-isd"
	size	20000
	owners_name	"gillogly"
site	id	"rand-isd"
	operating_system_name	"unix"
	machine_type	"pdp-11/45"
	guest_account_name	"netguest"
	guest_account_password	"netguest"
	known-user-set	("jjg," "rha," "rsg")
known_person	name	"gillogly"
	primary_site_id	"rand-isd"
	primary_directory	"jjg"
	primary_password	"whumpus"
	secondary_site_id	"cmu-10a"
	secondary_site_directory	"g250a12"
	secondary_site_password	"foo"

Figure 1—Examples of RITA object types, attributes, and values

Left-hand-side predicate clauses

IF: the name of the system is "unix"

IF: the name of the system is the name of the desired_system

IF: the name of the system is not known

IF: there is a response whose arrival_time is less than the max_expected_delay of the system

Right-hand-side action clauses

THEN: set the name of the system to "net access program"

THEN: set the valid_id_set of the remote_site to the id of every site whose id is known

THEN: deduce the guest_account_name of the remote_site

THEN: create a remote_site whose id is "cmu-10a"

THEN: receive the next line from the system_IO_pipe as the value of the response

THEN: send "Which rule do you wish to see" to the user

THEN: return success

Figure 2—Examples of RITA rule clauses

Such facilities as string manipulation are provided in the RITA system by a set of primitive functions which may be called in the predicates or actions of rules.

Monitors

We have found that different types of monitors are necessary for various specific tasks and situations, and that no one monitor type is sufficient for our purposes. For example, interactions with an external information system to handle routine protocols are best handled by a LHS-scan monitor (one which tests the pattern part, or left-hand side, of rules against the context to determine the next rule(s) to be applied), acting in what might be called a "stimulus-response" mode. On the other hand, it is sometimes necessary for an intelligent terminal agent to make deductions (e.g., about the most likely site on the ARPANET for a particular person to have a mailbox, given that person's attributes). Deductions are best made by a RHS (right-hand side, or action part) scan, goal-driven monitor. In this form of monitor operation, one specific item of information (an attribute of an object) is designated as a goal. The monitor seeks to execute rules whose RHS set that attribute's value. To execute those rules, their LHS must be true when tested against the current context. If any such LHS is not true due to the lack of information about the value of some other object's attribute, that attribute becomes a (sub)goal of the system. This process recursively forms a tree structure linking the applicable rules in the system, until one of three situations occurs: (1) a rule's LHS evaluates as true, so that its corresponding actions can be performed and no new subgoals are formed; (2) a rule's LHS evaluates as false, which also terminates that branch of the goal tree; (3) one or more attribute values are needed for the evaluation of an LHS, but no new subgoals can be formed because there are no other rules in the rule set whose action parts set those attribute values. In this latter case, the system queries the user for the values as a last resort. The system therefore tends to ask questions which "make sense" because it is following a particular chain of logic, and asks only those questions it needs to reach a particular goal, given the current state of its information base.

We have implemented several different monitors as part of the RITA system:

- LHS scan, with ordered rule set
- LHS scan, with unordered rule set
- RHS scan with backward-chaining (implicitly, a rule set is treated as unordered)

Nothing in our implementation precludes the development of other monitors if the need arises. The top-level monitor in a user agent uses a LHS scan with an unordered rule set; however, it is possible for the following action clause of some rule to be executed:

DEDUCE attribute OF object.

This clause triggers the operation of the RHS scan backward-chaining monitor with the goal of deducing the value of the named attribute. Upon completion of a deduction, control reverts to the action clause of the LHS scan rule following the DEDUCE clause, or if there are none, to the next applicable rule chosen by the LHS scan monitor. It is not possible to invoke explicitly an LHS scan monitor during a goal-directed deductive operation. A discussion of the differences between our goal-directed mode and that employed in the MYCIN system is contained in Reference 2.

System architecture

RITA has been designed as three cooperating modules, so that only the currently active module need reside in core during the operation of a user agent. Such modularity is aided by the facilities in the UNIX operating system for communication between separate processes.

The user interface module gives the user one or more windows within which text can be displayed, with the facilities of the Rand Editor (a CRT-based text editor with two-dimensional editing features) available for text manipulation within those windows. It allows creation of rule sets and contexts in a symbolic English-like form, and passes rules and commands to the monitor to allow user control over its operation.

The syntax module contains facilities for compiling symbolic form rules and data descriptions into an internal list-structure form. The monitor module accepts these "compiled" rules and has facilities for decompiling internal forms back into symbolic form upon request. It uses one of the available monitors to apply a rule set to a context, and emits trace information to a history file for use by diagnostic and tutorial facilities.

Advantages and disadvantages of production systems

Why were production systems chosen as the basis for the RITA system? The following advantages are often cited as accruing from their use. We have listed them in what we believe is an approximate decreasing order of importance for the particular application for which RITA was designed: namely, the construction of intelligent terminal agents.

1. Their explanatory capability.

Production system rules are intended to be modular chunks of knowledge and to represent primitive actions. Thus, explaining primitive acts should be as simple as stating the corresponding rule—all necessary contextual information should be included in the rule itself. Achieving such clear explanations, however, evidently strongly depends upon the extent to which the assumptions of modularity and explicit context are met.[4]

The interested reader is referred to the MYCIN literature for an excellent example of the degree of explanatory power that can be achieved through careful design and implementation.

2. Simple control structure.

Due to the simple control structure of production systems, especially of the LHS scan type, we can imagine the following type of instructions being nearly sufficient to introduce a user to the operation of his terminal:

> This terminal operates according to a set of rules. Whenever it finds a rule that is true, it applies that rule. If you want to know why it is asking you for some item of information, or why it took some action, type "why?" and it will show you the rules it followed in taking that action.

If, in addition, the rules themselves are in simple English so that they are directly readable by a user, then we believe he will find the operation of this device quite understandable. Although the user will of course not understand all the nuances of its operation, he is at least not bewildered at the start, and can add incrementally to his understanding with experience. The user must realize, however, during this initial introduction to the system that there are nuances and that he should not be overly complacent or trusting of system behavior.

A RHS scan backward-chaining system, although more complex in its control structure, can give rational explanations of its behavior in a manner that makes the flow of control among rules understandable.

3. Incremental addition of knowledge.

With proper design, production systems can allow gradual, incremental addition of knowledge and heuristics in a top-down manner. If the set of rules is unordered, then new rules can be added to the set without concern for their placement. A particularly appropriate time for the addition of new rules to a system is when the system, in a goal-oriented mode of operation, has asked a question of the user. A possible user response is to give the system a rule for determining that item of information from other information it has; upon receipt of that rule, the system will no longer ask that question, since it can now form a subgoal by backward-chaining through the new rule. In this manner gradual evolution of the behavior of the system takes place to meet the needs of the user in his possibly unique environment.

4. Trainability and learning.

Assume production rules are stated in a constrained syntax so that their meaning is understandable by machines, and that each rule is, to the extent possible, a "noninteracting chunk of knowledge or behavior." It then becomes possible for a computer program to create rules in the proper format and insert them into existing sets of rules to change the behavior of a production system. For examples of such adaptivity in production systems, see References 10 and 11.

There are also some disadvantages in the use of production systems. The two major ones are:

1. It can be difficult to code an operation in the form of a production system, particularly for goal-oriented rule sets. Considerable thought must be given to the choice of objects, attributes, and values by which a problem area is represented. (However, the problem of choosing a good data representation is certainly not unique to production systems; the problem lies more in trying to fit all applications into this particular procrustean bed.) One must also carefully choose certain attributes of objects to represent "state variables" which encode the state of a computation or deduction. The values of these state variables are tested by various rules to trigger their potential applicability. In this manner, production systems encode explicitly that which in ordinary high-level programming languages is implicit in the nesting of control statements. For example, a traditional nested control structure such as:

```
if A then
   if B then C
      else  if D then E; else;
   else G;
```

might be encoded in a production system in the following manner:

```
if A                      then state_1;
if state_1 and B          then C;
if state_1 and not B      then state_2;
if state_2 and D          then E;
if not A                  then G;
```

Such explicitness in a production system allows the desired relative autonomy of individual rules, but at the price of requiring the programmer to create names for many intermediate states of his process.

In the RITA system, we hope to overcome this disadvantage by having system experts create initial systems and user agents having general applicability. Individual users are expected, at least initially, to make only rather minor modifications and enhancements to the basic system. Therefore, the vocabulary and overall design of a user agent will be established, providing many guidelines and examples for the individual user.

2. Production systems are often inefficient. It is quite easy to design systems in which the pattern parts of hundreds of rules are tested against the data base before a successful match is found; it is also easy in goal-oriented systems to pursue lengthy chains of reasoning which are not useful.

Our design of the RITA system has not been significantly influenced by efficiency considerations. The simple user agents which have been constructed to date (e.g., for handling File Transfer Protocol interactions on the ARPANET) have required only 30 to 40 rules and are not inefficient. As more complex agents are constructed, we believe there are a number of monitor enhancements than can increase efficiency (e.g., through hash-coded lookup tables to aid in finding applicable rules) which can be added as the need arises.

EXAMPLE OF RITA OPERATION AS A NETWORK ACCESS AID

The following is an annotated transcript illustrating the operation of a RITA user agent designed to assist in file transfers from remote sites over the ARPANET. The agent consists of 42 rules and a context having 14 objects. Several representative rules from this user agent are shown in the middle of the transcript. Annotations are indented and enclosed in square brackets. User interactions with the agent are shown italicized.

```
% file-agent
```

["%" is the UNIX system prompt. An executable UNIX file named "file-agent" contains commands to start a RITA agent with the rules to be read from a file called "ftp. rules".]

```
UFE: V21 Dec. 16
PARSER: V22 Dec. 8
MON: 17 Dec 75
```

[These three header lines are displayed by various RITA processes as they initialize themselves, as an aid to system developers should trouble arise during this session.]

ftp.rules: [Displayed by RITA to show this rule set is being loaded; any syntax errors, etc. would be printed at this time.]

* *run;* ["*" is the RITA user front-end prompt; the "run" command starts the agent.]

[The following 6 questions are generated by RITA during its backward-chaining deduction process.]

What is the name of the current-file?

draft2

What is the site-name of the current-file?

sumex

What is the host-id of the current-file?

[The file is being retrieved from a site that the agent has never encountered before, so it checks that the site-name given is really the official host id for that site. As will be seen below, it doesn't need to be told this again in a subsequent session.]

sumex-aim

What is the account-name of the current-file?

kowalski

What is the password of the current-file?

qwerty

What is the man-number of the current-file?

why

[At this point, the user doesn't know what data is being asked for, so he uses the reserved word "why" to ask the agent for the chain of logic that led it to ask this question.]

That's what I was supposed to deduce.

What is the man-number of the current-file?

[This is RITA's answer if it has executed the action clause "DEDUCE man-number OF current-file" in some rule, but no goal-oriented rules are relevant in helping it deduce that; therefore, there is no deeper chain of logic that got it here. The user wants to better understand how the man-number is used by the agent, so he interrupts the agent's operation and returns to its command level.]

* *display all goals that test the man-number of the current-file;*

[This question lets him see all goal-oriented rules (denoted by a prefix "GOAL") that test that attribute in their "if", or predicate, part.]

```
GOAL not-known:
      IF: the man-number OF the current-file IS
            NOT KNOWN OR the man-number OF
            the current-file IS " "
    THEN: SET the file-preface OF the current-file
            TO " "
          & SET the file-suffix OF the current-file
            TO " ";
GOAL before:
      IF: nsubstr (0, 1, the man-number OF the cur-
            rent-file) IS "<"
    THEN: SET the file-preface OF the current-file TO
            the man-number OF the current-file
          & SET the file-suffix OF the current-file
            TO " ";
GOAL after.:
      IF: nsubstr (0, 1, the man-number OF the cur-
            rent-file) IS NOT "<"
    THEN: SET the file-suffix OF the current-file TO
            the man-number OF the current-file
          & SET the file-preface OF the current-file
            TO " ";
```

[Having seen the above rules, the user thinks the system doesn't handle TENEX systems correctly, and decides to give it a rule which will allow it to deduce the man-number of a file from the account-number on such systems. He therefore reinitializes the agent and types in a new rule at the agent's command level.]

* *restart;*

* *goal tenex*

```
    if: the type of the system is "tenex"
  then: set the man-number of the current-file to
          concat ("<", the account-name of the cur-
          rent-file, ">");
```

[He then re-runs the agent to see how it will act with this new rule added.]

```
* run;
What is the name of the current-file?
draft2
What is the site-name of the current-file?
sumex
What is the host-id of the current-file?
sumex-aim
What is the account-name of the current-file?
kowalski
What is the password of the current-file?
qwerty
What is the type of the system?
tenex
```

[Note that instead of asking for the man-number of the current-file, the system asked for the system's type instead, through use of the newly-added rule.]

```
Will now attempt to retrieve file <kowalski>draft2
from the sumex-aim system
Sent: ftp
```

[Lines prefixed with "Sent:" and "Received..." are generated by the agent during its operation to show its interactions with the remote system on behalf of the user. The agent could easily be modified to be less verbose.]

```
Received from net: Host:
Sent: sumex-aim
Received from ftp: Connections established.
300 SUMEX-AIM FTP Server 1.44.0.0—at SAT 3-
JAN-76 14:22-PST
>
```

[">" is the ftp prompt character]

```
Sent: user kowalski
Received from ftp: 330 User name accepted. Password,
please.
>
Sent: pass qwerty
Received from ftp: 230 Login completed.
>
Sent: retrieve <kowalski>draft 2 ftpdata
```

[Note that this RITA agent automatically places retrieved files in a local file named "ftpdata"; through the addition of one rule, the agent's behavior could be changed so that it requests a recipient file's name from the user.]

```
File has been transferred
Received from ftp: 255 SOCK 3276899843
250 ASCII retrieve of <KOWALSKI>DRAFT2.;1
started.
252 Transfer completed.
Sent: bye
Job finished
done
%
```

[All ARPANET File Transfer Protocol interactions necessary to retrieve the desired file were successfully accomplished automatically by the RITA agent.]

[The next day, the user wants to retrieve that same file again, since he just received an ARPANET message stating that he should look at some late revisions. This gives us an opportunity to see how much this particular RITA agent has learned from the previous session.]

```
% file-agent
UFE: V21 Dec. 16
PARSER: V22 Dec. 8
MON: 17 Dec 75
ftp.rules
* run;
What is the name of the current-file?
draft2
What is the type of the system?
tenex
Will now attempt to retrieve file <kowalski>draft2
from the sumex-aim system
```

[Note that once it received the file name "draft2", it assumed all other attributes about the file from information received during the current session. All of this adaptive behavior is handled by several RITA rules which build new rules and data objects as their actions, then save the revised rule set on a disk file.]

[To tune this agent's operation better, it should also be given a rule telling it that the type of a system doesn't change; once it knows the type of a remote system, it can assume that it will stay constant. With that information, the agent would stop asking for the type of previously encountered systems.]

```
Sent: ftp
Received from net: Host:
Sent: sumex-aim
Received from ftp: Connections established.
300 SUMEX-AIM FTP Server 1.44.0.0—at SAT
  3-JAN-76 14:26-PST
>
Sent: user kowalski
Received from ftp: 330 User name accepted. Password,
  please.
>
Sent: pass qwerty
Received from ftp: 230 Login completed.
>
Sent: retrieve <kowalski>draft2 ftpdata
File has been transferred
Received from ftp: 255 SOCK 3276899843
250 ASCII retrieve of <KOWALSKI>DRAFT2.;1
  started.
252 Transfer completed.
>
Sent: bye
Job finished
done
* exit;
exiting.
%
```

[The "exit" command terminates execution of the RITA system, and returns the user to the UNIX command level.]

[As a last example, we show this RITA agent retrieving a different file from the same site.]

```
% file-agent
UFE: V21 Dec. 16
PARSER: V22 Dec. 8
MON: 17 Dec. 75
ftp.rules
* run;
What is the name of the current-file?
draft3
What is the site-name of the current-file?
sumex
What is the type of the system?
tenex
Will now attempt to retrieve file <kowalski>draft3
  from the sumex-aim system
```

[Note that this time it asked for the site-name, but once it received a previously known response, it automatically retrieved other needed attributes about that site, such as its formal ARPANET host id. As mentioned above, the "type of system" question is unnecessary, and should be eliminated through the addition of another rule.]

```
Sent: ftp
. . .
```

[We omit the remainder of this session transcript, since it proceeds in the same manner as the interactive protocols shown above.]

CONCLUSION

This paper has discussed the design of the RITA system. Within RITA, user agents consisting of sets of production rules can be created to operate either autonomously or in interaction with a user. One major application of such user agents is to aid in interactions with computer networks and remote information systems. RITA is currently operational on a PDP 11/45 minicomputer running under the UNIX operating system. It occupies about 58K bytes of core, allocated among two separate processes. User agents comprising about 50 rules are now in use, and can, for example, handle ARPANET file transfer operations and interactions with the New York Times Information Bank.

This is, however, a report on work in progress. Some of the questions that remain unanswered at this stage of the research project are:

- Will a computer-naive user really be able to modify the operation of a user agent by adding or modifying rules? If so, how long a prior familiarization period is required?
- At what level of size or complexity of a user agent will speed of operation and efficiency become important considerations?
- What about system security? Knowledge about passwords, access keys, data formats, and account numbers for external systems might well reside within RITA user agents, making the intelligent terminal itself a valuable target for compromise. Even with physical security, such as restricting access to the room containing the terminal, often

there will still remain external communication paths to the machine that allow possible access to data. We need to understand more about the constraints which must be placed on access to intelligent terminals containing sensitive data in representative user environments.

We are encouraged by our initial experimentation in the use of production systems to represent heuristics governing intelligent terminal behavior. Their ability to provide explanations of that behavior, to be modified and incrementally extended by a user, and to operate in either a pattern-directed or goal-directed manner are all potentially valuable features. We believe the RITA system, whose design we have discussed here, provides a good testbed for the demonstration and evaluation of these intelligent terminal agent capabilities.

ACKNOWLEDGMENTS

Bob Greenberg designed and implemented many key features of RITA's user interface and syntax modules. The particular RITA agent whose operation created the transcript presented earlier was created by Dr. Don Waterman. The support and encouragement to date of our prior ARPA program manager, Dr. Craig Fields, and our current program manager, Steve Walker, are gratefully acknowledged.

REFERENCES

1. Anderson, Robert H., "Advanced Intelligent Terminals as a User's Network Interface," *Proceedings, IEEE COMPCON '75 Conference*, 9-11 Sept. 1975, Washington, D.C. (Also available as P-5445, The Rand Corporation, Santa Monica, California, June 1975).
2. Anderson, Robert H. and James J. Gillogly, "Rand Intelligent Terminal Agent (RITA): Design Philosophy," R-1809-ARPA, The Rand Corporation, Santa Monica, California, January 1976.
3. Ritchie, Dennis M. and Ken Thompson, "The UNIX Time-sharing System," *Communications of the ACM*, Vol. 17, No. 7, July 1974, pp. 365-375.
4. Davis, Randall and Jonathan King, "An Overview of Production Systems," STAN-CS-75-524, Computer Science Department, Stanford University, October 1975.
5. Shortliffe, E. H., S. G. Axline, B. G. Buchanan, T. C. Merigan and S. N. Cohen, "An Artificial Intelligence Program to Advise Physicians Regarding Antimicrobial Therapy," *Computers and Biomedical Research*, Vol. 6, 1973, pp. 544-560.
6. Shortliffe, E. H., S. G. Axline, B. G. Buchanan, and S. N. Cohen, "Design Considerations for a Program to Provide Consultations in Clinical Therapeutics," *Proc. 13th San Diego Biomedical Symposium*, February 4-6, 1974, pp. 311-319.
7. Shortliffe, E. H., "MYCIN: A Rule-based Computer Program for Advising Physicians Regarding Antimicrobial Therapy Selection," STAN-CS-74-465, Computer Science Department, Stanford University. October 1974. (Also available as Stanford Artificial Intelligence Laboratory Memo AIM-251.) A condensed version will be published as "MYCIN: Computer-based Medical Consultations" by American Elsevier Publishing Co., Inc. New York, 1976.
8. Shortliffe, E. H. and B. G. Buchanan, "A Model of Inexact Reasoning in Medicine," *Mathematical Biosciences*, Vol. 23, 1975, pp. 351-379.
9. Shortliffe, E. H., R. Davis, S. G. Axline, B. G. Buchanan, C. C. Green, and S. N. Cohen, "Computer-based Consultations in Clinical Therapeutics: Explanation and Rule-acquisition Capabilities of the MYCIN System," *Computers and Biomedical Research*, Vol. 8, 1975, pp. 303-320.
10. Waterman, D. A., "Generalization Learning Techniques for Automating the Learning of Heuristics," *Artificial Intelligence*, Vol. 1, No. 1 and 2, 1970, pp. 121-170.
11. Waterman, D. A., "Adaptive Production Systems," *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, 3-8 September 1975, pp. 296-303. Also available as CIP Working Paper 285, Psychology Department, Carnegie-Mellon University, Pittsburgh, Pa. December 1974.

Network interface systems—An evaluation by simulation*

by JOE B. WYATT and VINCENT I. POLLEY
Harvard University
Cambridge, Massachusetts

ABSTRACT

The availability of low-cost digital communication services has made it productive for computer users to remotely access multiple computer resources. This is a particularly sound strategy for accessing complex resources that are infeasible to relocate. At the same time, the availability of low-cost and reliable minicomputer systems which can accommodate sizable numbers of terminals has enabled a shift of some interactive programming and information processing to these small, standalone systems. Although these small systems are powerful and can be configured flexibly, there are a number of opportunities to expand the scope of an individual user by relating the small system directly to other host computers via a network. In accomplishing this feat, however, there are a bewildering number of technical and economic alternatives.

This paper addresses one of the issues surrounding the development of such systems namely the relative performance and cost characteristics of a minicomputer system in the role of both a standalone processor and a network interface. The methodology for the evaluation is stochastic simulation. The basic measurements for the base line simulations and model validation is a DEC PDP-11/45 computer system operating under the UNIX operating system in the Science Center at Harvard University. The paper describes the simulation methodology and the individual experiments in some detail. Several conclusions are drawn about the performance and cost characteristics of such systems and also about the use of simulation to evaluate these characteristics.

INTRODUCTION

One of the principal problems confronting a user when contemplating the use of a computer network is the variety and complexity of the system resources encountered. Conventions for system log-in, user command languages and terminal hardware interfaces vary from system to system. For someone who is neither skilled at nor interested in computer technology, it can be a frustrating confrontation. It has appeared that interfacing user terminals through a mini-computer system programmed to accommodate some of the interface functions might increase user productivity substantially. This possibility is particularly attractive since it has been demonstrated that small mini-computer, time-sharing systems are also highly cost effective for basic computer tasks; e.g., program development, management of small data bases and the like.[1]

In this paper a network interface system (NIS) is hypothesized in two configurations. The first configuration is a relatively large mini-computer which provides, in addition to the network interface functions, basic programming and language capabilities as well as file storage and retrieval facilities. The second NIS is a minimum configuration for handling network functions only.

The basis for both evaluations is the PDP-11/45 computer system operating under the UNIX operating system. Such a system is installed and operational at Harvard University. This system was used to develop the parameters for a simulation model and to validate the simulation results for the initial NIS configuration. The simulation model was developed using OSSL, a macro-level, stochastic language and processor developed for simulating operating computer systems.[2]

THE EXPERIMENTAL SYSTEM

The base case for the initial simulation model is the time-shared mini-computer facility located in the Science Center at Harvard University. This system is available for undergraduate teaching and research and provides basic hardware and software capability for several programming languages, information storage and retrieval systems, and a number of basic mathematical systems used in teaching programs.[3] The configuration consists of a PDP-11/45 processor with 120K words of executable memory (90K available to users jobs), a fixed head disc unit (used for swapping), two secondary storage disks available for user files, and two magnetic tape units. The system has available 33 ports for interactive user terminals (mostly 1200 baud CRT terminals) and a high-speed line printer for large vol-

* This work was supported by NSF Grant GJ40586.

ume hard copy output. It costs about $150,000. The system supports several language processors, a file system and editor, and a host of specialized processors operating under the UNIX operating system (supplied by Bell Laboratories).

The PDP-11/45 system first became operational in September of 1974, and was used primarily to teach the introductory computer science course at Harvard. The language used in this course was developed at Harvard and goes by the name of PPL (polymorphic programming language). The students using this language constituted over 60 percent of the system load during 1974-75. In addition to PPL there are also four other principal processors supported on the system. These include ECL (extensible coding language), another Harvard designed language, and also a lightly used BASIC compiler. The BASIC compiler functions very similarly to PPL and for simulation purposes was considered to be equivalent. An EDIT processor, which includes a file storage and retrieval system, is reasonably heavily used. Since the system is also used to teach a course which involves assembly level programming, the PDP-11/45 Assembler processor also receives some use, and is modeled in the ASEM submodule.

For the simulation of the basic functions of the system, a mix of typical terminal activities was determined from the set of processors above including an additional job to represent the execution of a compiled or assembled program. This TEST procedure simulates the execution of a job which accepts some user commands, executes a relatively large number of processor instructions, and produces a message back to the user terminal.

The load simulated was a very heavy one so that the dynamics of the PDP-11/45 system could be studied under some stress relative to the system scheduler. Data was collected from the system during a period of intensive use and used to verify the accuracy of the simulation model. The basic representation of the Harvard UNIX operating system was simulated by a submodel developed over a period of a year during which the structure and code of the system were closely examined and a relatively large number of measurements were made of system functions in an attempt to assure a valid model.

In addition to the fundamental processors described above, five functions for network interface were considered for the simulation model. These include NAM, a network access processor; FTP, a file transfer protocol; TRAN, a job control language translator; an accounting function processor; and an information storage and retrieval processor.* The latter two processors were treated in a special manner. The accounting functions were assumed to be absorbed in the "overhead" of the UNIX system (as indeed most are), since the system offers relatively complex capability for job accounting as implemented at Harvard. The information storage and retrieval functions which would be associated with obtaining network information are represented as EDIT jobs in the system (the EDIT jobs rely on the UNIX file system for storage, retrieval and processing). The former three network functions were simulated by discrete submodels.

The NAM is one of the most complex of the network submodels. It simulates the activities of a software machine extant at the National Bureau of Standards in Gaithersburg, Maryland.[4] The NAM is designed to effect log-on and subsystem access procedures for the user by assuming his duties in the necessary exchange of requests and information required while logging onto a network computer resource. NAM accepts a line of text from a user calling the NAM into operation. It then initiates communication with a remote host computer, exchanging between 6 and 18 lines of text in order to effect the requested access, and finally passes on to the user a "welcome aboard" missive from the same remote host. NAM also operates optionally as a character Store-and-Forward processor.

The submodel for the File Transfer Protocol (FTP) emulates the shipping of a word package from one host to another. The protocol being simulated is much like the ARPAnet FTP, wherein the data being shipped across the net is broken up into packets (here, of length 512 words), with receipt of a packet being acknowledged only after full error-detection software scans of it have finished. This FTP is different from ARPA's FTP in that the data shipped from Host A to Host B is stored at an intermediate machine (in this case at the **N**etwork **I**nterface **S**ystem) instead of travelling directly between the two machines. The NIS performs such functions as data transliteration and translation, coordination of device speeds (i.e., it buffers the transmission), and generally makes the whole procedure as transparent to the two hosts as possible. As far as they are concerned, they are outputting data to a standard I/O device.

The Translation submodel (TRAN) simulates a software package operational at Harvard's Office for Information Technology. This program effects translations of IBM's Job Control Language from one IBM system to another. For example, a job designed to be run on MIT's IBM 370/168 is passed through the Translator to acquire compatibility with the JCL conventions extant at Princeton University's IBM 360/91 Computer System. The TRAN Procedure simulates the table-driven translation of a medium-sized JCL deck, and is thought to be general enough in conceptualization to represent other types of simple translation, not merely JCL-type control cards.

These three Network functions, NAM, FTP, and TRAN, taken with the five previously discussed production jobs, make up the eight job types whose impact (in various combinations) upon the performance of

* The NAM model was developed from the programs and data supplied by the National Bureau of Standards Institute for Computer Sciences and Technology.

the PDP-11/45 UNIX computer will be the subject of the rest of this paper.

The OSSL developed simulation model consists of Procedures to represent the previously mentioned eight processes representing the eight basic system tasks, and one main Procedure (called SCHED) which simulates the actions of the UNIX Operating System. Procedure SCHED simulates the scheduling of system tasks, and coordinates the running of the other eight Procedures.

The UNIX method (simply stated) for determining which process shall be swapped into executable memory goes as follows:

(0) if a job is ready for swap-in, go to (2) else go to (1)

(1) enter a wait state and remain in it until signalled that a job is ready to be swapped in and then go to (2)

(2) find the job which has been swapped out the longest and which is runnable (i.e., it is waiting to use the processor) and then go to (3)

(3) if there is enough core available, swap the job found in step (2) into core and go to (1) else, go to (4)

(4) check to see if any jobs now in core are doing terminal I/O. If so, swap the first found out and go to (2) else, go to (5)

(5) if the job requesting swap-in has been out for 3 seconds or more, and if any job has been in core for more than 2 seconds, then swap the oldest such out and go to (2) else, go to (6)

(6) since either the oldest job on disc is less than 3 seconds old, or the oldest job in core is less than 2 seconds old, no job is eligible for swap-in. go to (1)

For simulation, procedure SCHED receives "requests" from the other eight procedures, asking that the jobs they represent be swapped into core (executable memory). It then steps through the above algorithm and, when it has finally "swapped the job in," notifies the relevant Procedure that its particular part of the simulation may continue.

THE SIMULATIONS

The subject system under analysis as the base case consists of the PDP-11/45 processor, 90K words of executable memory (available to user jobs), a fixed head disc operating at 250,000 words/sec transfer rate, and a secondary storage disk (available for user storage) operating at 150,000 words/sec with an 8 msec latency and a 25 msec seek time (the latter being an optimized 32 msec seek). The operating system is UNIX, and the job mix* is as follows:

five	PPL	jobs
two	ECL	jobs
three	EDIT	jobs
one	TEST	job
one	ASEM	job
one	NAM	job
one	FTP	job
one	TRAN	job

Each of these jobs is under the control of a hypothesized user (i.e., there are 15 people actively using the system) at his own 120 character-per-second CRT. Each is represented by an OSSL submodel. It should be noted that as each of the jobs in the standard job mix incurs completion, another of the same type is started up. Several "sensitivity" experiments were performed as baseline tests of the OSSL simulation model. The results of each sensitivity analysis from the simulations have been summarized, with two types of parameters presented. The first type includes internal *system performance* measurements, e.g., component scheduling and use measurements. The second type includes *use performance* measurements of response and throughput, e.g., the response times for individual interactions, and the completion rate for jobs. The description for each of the two types of parameters is shown in the following tables (Tables I and II).

Basic 11/45 system

After having run a basic set of models without the network interface system functions to validate the "stand alone system," the first new configuration was simulated with the network interface functions incorporated into the currently installed PDP-11/45 at the Harvard Science Center. The previously defined standard job mix was used in the base line simulation. In order to further exercise the base case in an evaluation of sensitivities, the system configuration was altered in several ways and simulated for the same time period with the identical job mix. Since the initial simulations indicated a processor intensive (and processor limited) system execution, the base line cases were tested on a configuration with four different processor configurations. The first was the basic PDP-11/45 processor, the second was a PDP-11/45 processor with Cache memory, the third was a PDP-11/70 processor with Cache memory, and the fourth was a dual PDP-11/45 processor system. The results of the base line simulations for the system performance and use performance parameters (previously described) are shown in Table III and Table IV.

One of the principal purposes of the four base line cases was to examine the sensitivities indicated by the model as the configuration was changed in order to determine the extensibility into further analysis of various network interface system configurations. It can be seen from the data in Table III that the performance of

* This mix (hereafter referred to as the "standard job mix") reflects a heavy load upon the system. Indeed, the system is functioning in its very worst case when supporting such a user distribution. However, such a load is a frequent occurrence at the end of the academic term.

TABLE I—Internal System Performance Parameters

CPU Busy —the percent of total time the processor has been active

CPU Rejection —the rate at which the processor has been requested but has been busy. The new request is then queued, and the queue residency time is the next figure cited (CPU Wait).

CPU Wait

mean —the average residency time of a queued processor request

σ —the standard deviation of the residency times

maximum —the maximum residency time of a queued processor request

CPU Wait Until

50% —the median residency time

66% —the time spent in queue by 66% of the requests

95% —same as above but for 95% of the requests

PRFR —The Processor Request Failure Rate (PRFR) is calculated as follows: of those requests for the processor which were queued, the percentage which were not met until 2 or more seconds had passed is multiplied by the fraction of requests for the processor which were queued. Thus, the PRFR reflects the chance that a random request for the processor will be kept waiting for more than 2 seconds. Since the UNIX scheduling algorithm allows a job to be swapped out of executable memory after it has been there for 2 seconds, the PRFR is a barometer reflecting about how many jobs will get into core, request the processor, and get flung from core before the request is met.

Swapping Disc Busy —the percent of total time the swapping disc has been active

Disk Busy —the percent of total time the disk storage unit has been active

Core Rejection —the percent of swap-in requests which could not be met the first time due to insufficient free core (required queueing)

Total Swap-in —the total number of jobs swapped-in

Average Core Life —the average time a job spent in core in any one swap

Average Core Used —the average amount of core in use by user jobs. (Also given as a percent of total available core.)

Note: In these reports, unless otherwise stated all times shall be in units of seconds, and all sizes will be in units of words (16 bits to each word).

TABLE II—External Use Performance Parameters

EDIT
mean —the timing distribution (mean, standard deviation and maximum) for completion of an edit command.
σ
max

MACRO
mean —the timing distribution (mean, standard deviation and maximum) for execution of a NAM remote system access call
σ
max

ASEM
ECL
EDIT
FTP
NAM
PPL
TEST
TRAN
—the counts reflecting the number of completions of the associated Procedure. If none have finished, the count will read zero. If none were started up in the first place (i.e., a particular simulation is not concerned with the relevant Procedure) then a dash (—) will appear.

the PDP 11/45, given the heavy work load for fifteen active terminals, was CPU (processor) limited. The CPU was busy almost 95 percent of the time and barely four requests in five could be met (the fifth had to be queued). The mean time in CPU queue was .87 seconds, with a standard deviation of 1.153 seconds and a maximum wait greater than five seconds. Since the UNIX operating system allows a process to reside in core normally for two seconds before being eligible to be swapped out, the results would indicate that 10 percent of the processes were eligible to be swapped out of core memory by the scheduler before they emerged from their wait for CPU service. None of the other performance parameters are particularly alarming. The swapping disc, core memory and disk storage are relatively modestly used. One might expect that their use was limited by the lack of processor capacity.

The use performance results from the simulation model are equally indicative of sluggish system response and throughput. For example, the mean response time for an EDIT command was in excess of five seconds with a standard deviation of over five seconds as well. The maximum recorded wait was 22 seconds. The NAM MACRO response is even more indicative of the processor bottleneck since MACRO requires the execution of a substantial number of instructions. The mean response time for MACRO was 196.7 seconds with a standard deviation of 56.1. The throughput counts for the assembler and the network interface sys-

TABLE III—Internal System Performance

System Performance	11/45	11/45 CACHE	11/70 CACHE	2 x 11/45
CPU Busy	94.9%	91.6%	88.0%	54.3 & 35.3
CPU Rejection	18.4%	16.9%	14.4%	none
CPU Wait—mean	0.87	0.48	0.23	no wait
—σ	1.153	0.64	0.35	—
—max.	>5.0	5.0	3.2	—
CPU Wait—until 50%	0.6	0.2	0.15	no wait
66%	1.0	0.6	0.25	—
95%	3.4	1.8	0.60	—
Core Rejection	58%	56%	55%	54%
Total Swap-in	1181	1317	1328	1448
Average Core Life	.507	.455	.452	.415
Average Core Used	89%	87.8%	—	84.2%
Swapping Disc Busy	29.8%	34.8%	39.9%	37.4%
Disk Busy	14.2%	23.1%	31.6%	35.5%
PRFR	10.1%	7.6%	3.6%	0

TABLE IV—Use Performance

		11/45	11/70	
Use Performance	11/45	CACHE	CACHE	2×11/45
EDIT—mean response	5.61	3.47	2.27	2.46
—σ	5.12	2.87	1.87	2.92
—max.	22.	12.	10.0	16.0
MACRO (NAM)—mean response	196.7	192.	92.9	40.5
—σ	56.1	45.	47.5	9.8
—max.	—	—	190	50
ASEM	7	11	17	17
FTP	41	69	116	247
NAM	2	3	5	7
TRAN	17	28	38	46

Note: The ECL, PPL, EDIT and TEST jobs are of sufficient length (5 to 15 minutes each) that none complete in the 10 minute base line simulations. A subsequent simulation of longer duration demonstrates throughput for these jobs.

tem jobs indicate a relatively modest completion rate for these functions.

From an examination of the results of the initial simulation it appears that the configuration is sensitive to the availability of processor bandwidth and possibly the amount of available executable memory (since the average core used was 89 percent). There is also the hint that prioritizing the scheduling algorithm in some different manner might affect system performance and use performance for a particular type of job.

11/45 cache processor

The relatively clear indication for a subsequent sensitivity run is to improve the processor bandwidth. As a result, the second of the base line simulations incorporated the use of a Cache memory added to the PDP-11/45 processor. The results from the effective improvement in the processor capacity can be observed from the results in Tables III and IV. The processor busy time has dropped from 95 percent to 92 percent. The processor wait time has been approximately halved with the standard deviation dropping from 1.15 to .64. It is also indicated that for half of the processes the residency time in the CPU wait queue dropped from .6 to .2 seconds. The improvement in CPU bandwidth has also resulted in more efficient use of the swapping disc and the risk storage unit (increased 5 percent and 9 percent respectively). The PRFR has also sharply reduced as expected.

Use performance of the system with the Cache has improved dramatically. For example, the mean EDIT response time has dropped from 5.61 seconds to 3.47 seconds (standard deviation dropped from 5.12 to 2.87) and the maximum dropped from 22 to 12 seconds. Throughput improvements average around 60 percent as characterized by the FTP job turnaround, which increased from 41 completions to 69 completions.

11/70 processor with cache

The indications from the prior simulation of the system with the 11/45 processor and Cache supported the notion that the CPU was and remains the limiting component. A third simulation replaced the 11/45 processor and Cache with an 11/70 processor and Cache leaving all other characteristics of configuration and workload constant.

This change also significantly improved the performance of the system. Throughput improved by about 60 percent and interactive response times also dropped. For example, the mean EDIT response was reduced to 2.27 seconds (σ of 1.87) and the maximum to 10.0. The NAM MACRO response improved from a mean of 192.0 seconds to 92.9 seconds. Internal system performance also reflected the improvement with CPU Busy time dropping from 91.6 percent to 88.0 percent and CPU queuing appreciably reduced. Memory swap-ins continued to increase for the period and both the swapping disc and the disk storage unit busy times jumped appreciably.

Dual 11/45 processors

The final sensitivity test in the baseline series accommodated dual 11/45 Processors to replace the single processor configuration. In this case, processor queuing was eliminated but some response times showed no gain over the uniprocessor 11/70 Cache system, reflecting the mixed blessing of two slower processors versus one faster processor in a processor intensive environment. In general, however, both use performance and internal performance were improved over the fastest uniprocessor configuration. It should be noted that no allowance was made for any increase in operating system overhead which might result from the dual processor configuration. The UNIX system used at Harvard does not accommodate other than a uniprocessor configuration and estimates vary about the consequences of a multiprocessor modification to UNIX.

Base line simulation conclusions

The four base line simulation experiments bolstered faith in the capability of the model to represent the

NIS system. The isolation of the processor as the constraining component in the system and the demonstration of the sensitivity of the model to improvements in processor performance seemed in keeping with expectations. Moreover, the subtle changes in the internal system performance of the other major components, e.g., executable memory, swapping disc, and disk storage, increased confidence in the model as well. Use performance data was also within predicted limits although priority modifications in the scheduler might have altered the performance of individual processors substantially.

It was particularly interesting in the simulation to note the continual improvement of the uniprocessor configurations as processor performance improved followed by the somewhat discontinuous improvement of the dual processor configuration which utilized the slowest processor. In particular, although overall performance indicated substantial improvement for the dual processor configuration, some individual processes did not perform as well on the dual processor configuration as on the fastest uniprocessor. Although in retrospect the result was not surprising, the ability of the model to accurately reflect the mixed blessing of the dual processor configuration supported the notion that the model was sufficient for more useful analyses.

TABLE V—Relative Effect on Internal Performance

	Fifteen Terminal (Standalone & NIS)*	Twelve Terminal (Standalone)	Fifteen Terminal (Standalone)
CPU Busy	88.0%	93.5%	92.76%
CPU Rejection	14.4%	59.5%	69%
CPU Wait—mean	.23	.40	.45
—σ	.35	.54	.55
—max.	3.2	5.0	4.2
CPU Wait-until 50%	0.15	.30	0.3
66%	0.25	.39	0.4
95%	0.60	1.5	1.6
Core Rejection	55%	47%	51.6%
Total Swap-in	1328	500	744
Average Core Life	.452	1.2	.807
Swapping Disc Busy	39.9%	11.92	16.14
Disk Busy	31.6%	3.82	3.53
PRFR	3.6%	5.7%	9.7%

* From Table III for comparison.

WORKLOAD SENSITIVITY ANALYSIS

The second part of the simulation experiment dealt with the evaluation of the relative effect of network interface functions combined with basic functions on the standalone processor. Specifically it was of some interest to learn whether the network interface functions had more or less impact on system performance than the basic "standalone" processors. More specifically, it seemed meaningful to understand whether the addition of n terminals to engage in network access activity would have more or less effect on the PDP-11 UNIX configuration than the addition of the same number of terminals for standalone tasks involving PPL, ECL, BASIC, and the like. As a result, several simulation experiments were conducted to attempt to evaluate the sensitivity of the model to these alterations in workload. For these experiments the PDP-11/70 Cache configuration was selected. The use of this configuration rather than any of the others provides a level of processor power sufficient to make either an increase or decrease in processor demand more obvious and more realistic.

Twelve terminal standalone system

The first simulation consisted of the PDP-11/70 Cache configuration with the job mix altered to remove the network interface functions. In other words, all use of NAM, FTP and TRAN was eliminated from the workload. In the first case this was accomplished by simply reducing the number of terminals from 15 to 12, removing those three that were engaged in the network interface functions.

The effect of this change in workload upon internal performance is shown in Table V. It can be seen that there is a general shift toward more processor-intensive use of the configuration and less efficient use of both the swapping disc and the disk storage units. Even though the total number of active terminals has dropped from 15 to 12, the processor has jumped from 88 percent to 93.5 percent, and the processor rejection rate (percent of requests which must queue for processor use) has soared from 14.4 percent to 59.5 percent. Other processor performance parameters, including the queue waiting statistics, further imply a relative increase in processor demand. On the other hand, the rejection rate for memory use has actually dropped from 55 percent to 47 percent. The total number of swaps has reduced dramatically and the average life of a process in memory has more than doubled. At the other end of the spectrum the swapping disc busy time has dropped from about 40 percent to 12 percent, and disk storage unit use has dropped precipitously to only slightly over 10 percent of its former use level.

From the perspective of system utilization, the results are limited but significant (see Table VI). The interactive response time as measured by the EDIT processor has remained relatively constant. The mean response time for the EDIT is at about 2.3 seconds with a standard deviation of about 1.9 seconds, for both simulations. The maximum response time has increased slightly for the non-network configuration. Throughput as indicated by the assembler has increased only slightly—from 17 to 19 jobs.

It can be concluded from this initial experiment that removal of the network interface functions slightly im-

TABLE VI—Relative Effect on Use Performance

	Fifteen Terminal (Standalone & Nis)*	Twelve Terminal (Standalone)	Fifteen Terminal (Standalone)
EDIT—mean response	2.27	2.3	2.59
—σ	1.87	1.9	1.81
—max.	10.0	12.0	8.0
ASEM	17	19	17

* From Table IV for comparison.

proves the throughput of the configuration for the other jobs but does not substantially alter the response time for interactive tasks. From the internal performance data, however, it is obvious that the system is unbalanced in the sense that the processor is intensely busy while other critical parts of the configuration, including executable memory, the swapping disc, and disk storage units, are performing well below their most efficient levels. Hence, the first indication would be that the network interface functions fit into the workload environment, for the most part consuming resources that are otherwise not critical.

Fifteen terminal standalone system

A second experiment to measure the relative effect of the network interface functions was conducted with a fifteen terminal system. In this case, the network interface functions NAM, FTP, and TRAN were replaced by standalone jobs representing PPL, EDIT, and TEST. This configuration brought the number of terminals up to equal the combined configuration tested first, but eliminated any network access functions. In this way, an indication of the relative effect of the network interface functions could be ascertained directly.

A summary of the results is shown in Table V for internal performance and Table VI for use performance. It can be observed that the use performance varied only slightly from the other 15 terminal configuration. The EDIT interactive response indicated both a slight change in the direction of improvement (maximum response dropped from 10 seconds in the combined system to 8 seconds) and a slight degradation (mean response time rose from 2.27 seconds to 2.59 seconds). Morcover, the throughput count for the assembler (ASEM) was the same. As a result, the relative effect of the network interface functions on system use performance is negligible.

The internal performance data indicates some change. The CPU busy rate rose from 88 to 93 percent, and the rejection rate soared to 69 percent from 14 percent. In addition, the mean wait time for the processor ready queue doubled, the standard deviation increased from .35 to .55 and the maximum increased by about one third. The use of executable memory became less contentious, however. The core rejection rate dropped slightly, the total number of swaps was cut almost in half and the average core life jumped from .45 seconds to .81 seconds. Most notably, however, the uses of the swapping disc and disk storage were only about forty percent and eleven percent respectively of the combined configuration! The clear indication from the internal performance data is that the network interface functions tend to balance the use of major system components more efficiently, reducing processor congestion significantly and using the idle resources of the swapping disc and the disk storage unit. Thus, the relative impact of the network interface functions appears to be at worst breakeven (use performance) and at best favorable (internal performance).

TABLE VII—Incremental Effect on Internal Performance

	Basic+3* 11/70 CACHE	Basic+3+NIS 11/70 CACHE
CPU Busy	92.76%	89.8%
CPU Rejection	69%	20.3%
CPU Wait—mean	.45	.40
—	.55	.46
—max.	4.2	3.4
CPU Wait—until 50%	0.3	.38
66%	0.4	.44
95%	1.6	1.4
Core Rejection	51.6%	56.8%
Total Swap-in	744	1283
Average Core Life	.807	.467
Swapping Disc Busy	16.14	27.3
Disk Busy	3.53	39.7%
PRFR	9.7%	6.1%

* From Table V for comparison.

Eighteen terminal combined system

The third and final experiment in this series was to determine the *incremental* effect of the network interface functions. This experiment was performed by simply adding three terminals with the three network interface functions to the fifteen terminal standalone configuration previously simulated. The results of this experiment are shown in Table VII and VIII. From Table VIII it can be observed that the throughput performance of the system was reduced only slightly for

TABLE VIII—Incremental Effect on Use Performance

	Basic+3* 11/70 CACHE	Basic+3+NIS 11/70 CACHE
EDIT—mean	2.59	2.3
—σ	1.81	1.6
—max.	8.0	8.0
ASEM	17	16
FTP	—	94
TRAN	—	34

* From Table VI for comparison.

the assembler while 94 FTP transactions were performed and 34 program translations (TRAN) were performed. The EDIT response times did not significantly change. Hence it would appear that the network interface functions were performed at a minute incremental expense. From the internal performance in Table VII it can be seen that the network interface functions place almost all of the incremental load on the disk storage unit with a noticeable increase on the swapping disc and executable memory use as well. Since the network interface functions are significantly less processor intensive, the average use of the processor has dropped modestly and the processor rejection rate has dropped dramatically (to 1/3 of its former level).

Conclusions about the combined NIS and standalone system

It appears that the network interface functions mesh well with the characteristics of the standalone UNIX system. On a relative basis, replacement of standalone functions with network interface functions tended to balance the use of the major components of the computer system configuration more evenly, thus reducing the processor intensivity. This would indicate that the configuration becomes relatively much less congested in the case where some terminals are engaged in network activities. On an incremental basis it appears that a few additional terminals could be added for network access to a standalone configuration with a very modest impact on the standalone functions. In addition, the incremental effect would be a more efficient use of all of the components in the system. Hence, insofar as system performance is concerned the network interface function can be integrated into the PDP-11 UNIX system at a negligible sacrifice in performance of the standalone functions and with substantial improvement in the relative efficiency of the major components in the system. This result is, of course, more useful with the assumption that the Harvard UNIX system is typical of other standalone mini-computer system configurations.

NETWORK INTERFACE SYSTEM PERFORMANCE

The third and final analysis of the investigation deals with the development of a mini-computer configuration exclusively for network interface functions. The objective is to provide adequate performance capability for network access alone at a minimum cost. For this analysis both the job mix and the computer system configuration were altered substantially. The job mix represented was felt to be typical of an individual user accessing a computer network resource via the facilities of the network interface system.

The essential flow goes as follows:

(1) A user, having logged on to the NIS system, enters the editor (EDIT) and edits a file of length 3K words, invoking 20 to 30 edit commands.

(2) Having prepared his file, he next uses the NAM to open a connection and log him in to a remote host.

(3) Ready to send his file (constructed in step 1) across the network to the remote host, he first must make its "JCL" compatible with the receiving institution's conventions. He thus translates it using TRAN.

(4) With the file ready to go and the network connections complete, he uses a File Transfer Protocol (FTP) to ship the translated file over the net.

(5) With his program ready to run on the remote host, he now uses the NAM, in its character store-and-forward mode, to send from 6 to 12 lines of text (receiving the same on a one-for-one exchange basis) to the remote host, setting the job up for running, and finally initiating execution by the remote host.

(6) Execution completed (in zero elapsed time for purposes of this analysis) on the remote host, the user uses FTP to ship his results home.

(7) And finally, now that he's finished with the remote host, he uses the NAM to close his network connections.

This seven-step terminal session is simulated for each terminal by the sequential execution of the procedures indicated with a variety of use parameters. Upon completion of the seven-step sequence, it is reinitiated for each terminal. The sequence is started at a random point for each terminal in order to more quickly reach steady state in the simulation.

Four hardware configurations are examined relative to the specified workload. Each configuration is simulated for a period of thirty minutes operation. The basic hardware configuration includes a PDP-11/20 processing unit (to replace the prior PDP-11/70 and 11/45 processing units) and a reduced executable memory configuration. The same swapping disc and disk storage devices from previous simulations are used. The configurations remain constant for the four simulation experiments except for the size of executable memory and the number of active terminals on the system. The operating system simulated was UNIX even though the PDP-11/20 processing unit has a slightly smaller instruction set which would require some modification to UNIX should it actually be used. In the four simulations, the following combinations of active terminals and executable memory were examined:

(1) Ten terminals and 20K executable memory.

(2) Fifteen terminals and 20K executable memory.

(3) Fifteen terminals and 32K executable memory.

(4) Eighteen terminals and 32K executable memory.

The ten terminal 20K system was chosen as the base line configuration. It was hypothesized from the early experiments that the PDP-11/20 processor, which is approximately one-fourth the speed of the PDP-11/70 with Cache memory, would be adequate to support a ten terminal configuration given the seven-step sequence of operations defined. The 20K word memory was estimated in size from the memory use statistics gleaned from prior simulations for the network interface function processors. Since there was some feeling that the availability of executable memory might be more sensitive in the NIS system than for the prior combined systems, additional statistics were collected from the simulation model for the queueing relative to the availability of executable memory. These statistics called "core wait" (mean, standard deviation and maximum) and "core wait until" (50 percent, 66 percent, 95 percent) are analogous to the same statistics collected for the central processing unit in prior simulations.

Base line NIS system (ten terminals and 20K memory)

The summarized results for the four simulation experiments are shown in Tables IX and X. Table IX indicates the internal performance of the NIS and Table X indicates use performance. From the use performance statistics it appears that the base line system is performing within reasonable limits relative to system use. The response time for the EDIT processor is about 3 seconds with a standard deviation of slightly over 2 seconds. The MACRO NAM processor response time is a mean of 31 seconds and a standard deviation of about 11 seconds (a performance superior to any of the prior configurations). Throughput for the base line configuration appears to be respectable although it cannot be directly compared to the prior combined system simulations because of the difference in the characterization of the network interface system workload.

TABLE IX—NIS Internal Performance

	10 Term	15 Term	15 Term	18 Term
	20K	20K	32K	32K
CPU Busy	52.7%	72.42%	74.7%	86.9%
CPU Rejection	19.4%	60.5%	45.8%	63%
CPU Wait—mean	.03	.019	.029	.043
—σ	.18	.18	.23	.332
—max.	4.0	>5.	>5.	>5.
Core Rejection	53.8%	61%	54.2%	55%
Core Wait—mean	.40	.91	.361	.45
—σ	1.15	1.62	.94	1.05
—max.	>5.	>5.	>4.5	>4.5
Core Wait—Until 50%	.15	.20	.15	.15
66%	.19	.75	.20	.20
95%	2.75	4.0	2.0	2.5
Total Swap-in	484	1158	596	918
Average Core Life	3.72	1.55	3.02	1.96
Swapping Disc Busy	1.45	3.41%	1.7%	2.5%
Disk Busy	4.16	6.22%	5.4%	5.7%
PRFR	0	.24%	0	.485%

The internal system performance for the NIS indicates that the processor is only about 50 percent busy with a very low request rejection rate and a very minimal amount of processor queueing. The rejection rate for initial requests for executable memory is about 50 percent, however the mean wait for executable memory is relatively high with a mean of .4 seconds and a standard deviation of 1.15 seconds. The additional statistics for executable memory use indicate that 2/3 of the queued requests were processed within .19 seconds and 95 percent were processed within 2.75 seconds, still relatively long waits for executable memory in a computer system. Offsetting the queueing is the indication that the average core life of a process once swapped into memory is 3.72 seconds. Both the swapping disc and the disk storage are trivially active, indicating about 1 percent and 4 percent busy respectively.

Fifteen terminal and 20K memory NIS system

In the second experiment the number of terminals was increased by 50 percent to a total of 15 with the remainder of the configuration constant. As a result of this increase the throughput of the system increased about 33 percent ranging from 27 percent improvement for the edit and 42 percent improvement for the NAM. This improvement in throughput was accomplished, however, at some sacrifice in interactive response time. The mean response to EDIT transactions increased from 3 seconds to over 4 seconds and the mean response for MACRO increased from 31 seconds to 49 seconds. These responses are within reasonable limits, but are approaching that state which the user might call "sluggish."

An examination of the internal performance data indicates that the processor is substantially more busy, reaching 72 percent with a rejection rate leaping from

TABLE X—NIS Use Performance

	10 Term	15 Term	15 Term	18 Term
	20K	20K	32K	32K
EDIT—mean	3.04	4.39	4.34	6.23
—σ	2.36	3.63	3.73	5.4
—max.	14.	18.	20.	28.0
MACRO (NAM)—mean	31.	49.2	42.9	52.6
—σ	10.7	18.9	18.2	21.7
—max.	60.	100.	90.	120.
EDIT	11	14	12	9
FTP	18	24	22	19
NAM	26	37	30	35
TRAN	9	12	10	11

19 percent to 60.5 percent for initial processor service requests. Queueing times, however, have not substantially increased for the processor. The availability of executable memory appears reasonably critical with a rejection rate jumping to 61 percent from about 54 percent and the mean wait in queue more than doubling. In addition, the total number of swap-ins more than doubled to 1158 from 484 and the average core life dropped from 3.72 seconds down to 1.55 seconds. It is significant to note, however, that the swapping disc and the disk storage unit continue to be hardly taxed at all with busy times of 3.4 percent and 6.2 percent respectively.

Fifteen terminal and 32K memoryNIS system

Since the second experiment which held the configuration constant and increased the number of terminals to 15 seemed to indicate that executable memory was in short supply, the third experiment held the number of terminals to 15 and increased executable memory to 32K words. This resulted in almost negligible change in use performance. Response times stayed about the same as the second configuration for the EDIT and showed very slight improvement for the MACRO processor. The throughput statistics remained virtually level for each of the four processors (with the exception of NAM which showed a somewhat anomalous drop due to the sequence in which the 15 different seven-set processes completed). In terms of internal performance the processor continued busy (74.7 percent) and the processor wait in queue times increased slightly.

The executable memory statistics clearly reflect the improvement concomitant with greater memory size. The rejection rate dropped from 61 percent to 54.2 percent and the queueing times dropped as well with the mean reduced to .361 seconds from .91 seconds. Total swapping was also cut approximately in half and average core life improved back to above 3 seconds from 1.5 seconds. One might conclude, however, on the basis of the use performance, that the investment in the additional 12K words of executable memory was not returned in visible performance to the user.

Eighteen terminal and 32K memory NIS system

As a final sensitivity test, the configuration of the system was again held constant and the number of terminals increased to 18 from the 32K word memory. The increase in terminals from 10 to 18 for the PDP-11/20 processor begins to show symptoms of a debilitating nature in this experiment. The processor busy time has increased to about 87 percent and the rejection rate has jumped up to 63 percent. Queueing times for the processor are at an all-time high mean of 43 milliseconds with a standard deviation of 332 milliseconds. As might be expected, the executable memory statistics are approximately the same for memory queueing but the number of swap-ins has increased from 596 to 918 and the average core life has dropped from over 3 seconds to 1.96 seconds. Hence the contention for the central processing unit has affected the use of executable memory to some degree.

Conclusions about the NIS system

For some of the network interface functions, the processor is a limiting factor as the number of jobs increase. For example, the EDIT processor is adversely affected as the number of terminals on the system increase and the processor remains constant. On the other hand, the NAM processor appears considerably less affected by the availability of processor power, although some constraining does occur.

It is apparent from all of these results, however, that a relatively small mini-computer configuration can accommodate a sizeable number of terminals performing network interface functions. For example, the configuration with 15 terminals operating on the 20K word PDP-11/20 system provides a superior use performance. There is also a reasonable balance of internal performance activity between the processing unit and executable memory. In all of the cases, however, the swapping disc and disk storage units appear excessively powerful for the need. It is reasonable to expect that the units simulated could be replaced by others, somewhat less sophisticated (and less expensive). Should swapping become a problem, it has been demonstrated that modest increases in executable memory size can reduce the amount of swapping and hence the dependence on a high performance swapping device. In addition, the range of performance characteristics for disk storage units available for mini-computers allows for a great deal of configuration "tuning."

These two hypotheses were in fact tested by simulating the substitution of a single, relatively slow disk storage unit for both the swapping disc and the disk storage device. The replacement disk storage unit contains two removable "packs" with capacity of three million bytes each operating through a single disk control unit. The essential performance characteristics of the replacement disk unit are:

Average Seek Time	—35 milliseconds
Average Latency	—12.5 milliseconds
Transfer Rate	—2.5 million bits/second

The disk replacement was the only alteration to the fifteen terminal, 20K executable memory system previously simulated (Tables IX and X).

The result confirmed the hypothesis that the lower performance disk storage unit could replace the separate swapping disc and disk storage units with modest sacrifice in performance (Tables XI and XII). As hypothesized, the addition of 12K memory reduced the

TABLE XI—Single (Slow) Disk NIS Internal Performance

	Fifteen Terminals (20K) Swapping Disc/ Storage Disk*	Fifteen Terminals (20K) Slow Disk	Fifteen Terminals (32K) Slow Disk
CPU Busy	72.42%	65.36%	73.6%
CPU Rejection	60.5%	55.1%	42%
CPU Wait—μ	.019	.031	.037
σ	.18	.23	.25
Max.	>5.	4.8	4.8
CPU Wait Until 50%			
66%			
95%		<.2	<.2
Core Rejection	61%	61.8%	56%
Core Wait—μ	.91		.717
σ	1.62		1.104
Max.	>5.		4.
Core Wait Until 50%	.20		.18
66%	.75		.25
95%	4.0		3.0
Total Swapin	1158	1040	675
Average Core Life	1.55	1.73	2.67
Swapping Disc Busy	3.41%	—	—
Disk Busy	6.22%	12.43%	12.98%
PRFR	.24%	.253%	.25%

* From Table IX for comparison.

TABLE XII—Single (Slow) Disk NIS Use Performance

	Fifteen Terminals (20K) Swapping Disc/ Storage Disk*	Fifteen Terminals (20K) Slow Disk	Fifteen Terminals (32K) Slow Disk
EDIT—mean	4.39	4.60	4.39
—σ	3.63	3.95	3.46
—max.	18.	20.0	16.
MACRO (NAM)—mean	49.2	59.57	50.9
—σ	18.9	21.0	13.6
—max.	100.	110.	80.0
EDIT	14	9	12
FTP	24	16	20
NAM	37	18	25
TRAN	12	6	8

* From Table X for comparison.

swapping considerably. The use performance statistics of the system would indicate that a single disk with lower total access time might completely restore the performance level. From this result, the configuration for the NIS system seems relatively well balanced to workload. It is comforting to know, however, that a variety of mini-computer system components (processors, Cache memories, executable memories, disks, etc.) are currently available at relatively low cost for incrementally configuring such systems.

CONCLUSIONS

On the basis of current computer system pricing, it is clear that a computer system configuration approximating that of the final network interface systems simulated could be purchased for under $40,000. This would include 32K word executable memory, floating point arithmetic, a 6 million byte disk storage unit and controller, a processor equal or more powerful than the PDP-11/20, and a communications interface controller and line controller for 16 terminals. The simulation analyses indicate that this would provide a system with adequate performance capability for network interface only, for standalone use only, or for a combination depending on the specific character of the workload mix. In fact, all of the simulations have been conducted with very intensive workload characteristics and have assumed that all of the terminals simulated were simultaneously heavily active. Hence, it seems likely that local use patterns for such systems would be less demanding than those simulated. Moreover, it is relatively easy to visualize the price of small computer systems on the order of those simulated to continue to drop dramatically.

If one amortizes the purchase price of such a system over a period of three years, the incremental cost per terminal for the network interface system and substantial standalone capability already comes to less than $1000. It might reasonably be expected that this investment could be recouped several times over in reduced networking costs (connect time, etc.) and in more effective use of the human user's time. Considering the capability promised by a combination of the small standalone computer system in conjunction with a hierarchical interface to a variety of network host computers, it seems likely that such an investment might be wise.

REFERENCES

1. Marienthal, Louis B., "Small Computers for Small Businesses," *Datamation*, June, 1975.
2. Wyatt, Joe B., "Computer Systems: Simulation," Chapter 19, *The Information Systems Handbook*, Editors: F. Warren McFarlan and Richard L. Nolan, Dow Jones-Irwin, Inc., 1975.
3. The Harvard/Radcliffe Student Time-sharing System Terminal Users Guide, 1st edition, September 10, 1974, Center for Research in Computing Technology, Harvard University.
4. Rosenthal, Robert, "Accessing On-line Network Resources With A Network Access Machine," *IEEE Intercon 1975*, IEEE, New York, 1975.

An overview of the distributed computer network*

by DAVID L. MILLS
University of Maryland
College Park, Maryland

ABSTRACT

The Distributed Computer Network (DCN) is a resource-sharing computer network which includes a number of DEC PDP11 computers. The DCN supports a number of processes in a multiprogrammed virtual environment. Processes can communicate with each other and interface with this environment in a manner which is independent of their residence within a particular computer. Resources such as processors, devices and storage media can be remotely accessed and shared so as to provide increased reliability, flexibility and system utilization.

The DCN now supports several programming languages and applications packages. Programming languages such as SIMPL, LISP, BASIC and others, along with an extensive library of interactive graphics procedures, can be executed in processes which take full advantage of the distributed architecture of the network. Many of the components of the Disk Operating System (DOS) for the PDP11 can be executed in a special emulator-type virtual process now being constructed for this purpose. In this manner the PDP-11 assembler, FORTRAN compiler and various system utilities can be supported in the network environment. In cases which exceed the processing power of the network, connections are available to two large Univac 1100-series machines.

INTRODUCTION

A resource-sharing computer network is a collection of computers and communication facilities in which resources resident on one computer can be accessed remotely from another.[9] A distributed computer network is a resource-sharing network in which resources are managed on a global scale so that files and processes can migrate throughout the network as determined by user requirements and the current system loading. In distributed networks individual users need not necessarily be aware of the physical location of their resources.

The Distributed Computer Network (DCN) project started at the University of Maryland in 1973 with a single PDP 11/45 which could be connected to either a dual-processor Univac 1108 or a single-processor Univac 1106. The PDP11 was to be used to develop a virtual-processor based operating system suitable for operation on a number of PDP11 computers. The 1106 and 1108 (hereafter referred to collectively as the 1100) were to be used for data-base residence and as a computational resource. Other papers and reports[11,12,14–18,25] have described original design objectives for the DCN and the operating systems developed for it.

The DCN has since grown to include several PDP11 computers and peripherals interconnected by a variety of interprocessor links and data communication devices. The operating system which controls the resources of the DCN exists in two versions. One of these, called the Virtual Operating System (VOS), supports virtual-processor emulation of standard PDP11 programs and requires the storage management features of the PDP11/45 or the PDP11/70. The other, called the Basic Operating System (BOS), supports special-purpose dedicated applications in PDP11 models without these features, including the GT40 Display Processor and the LSI-11 Microprocessor.

Design objectives

The DCN was originally intended as a research tool for the development and evaluation of resource allocation and management techniques suited to the distributed environment. Although intended to be used by students and faculty, it was not anticipated that the DCN would replace other operating systems like RSX11[4] and UNIX[21] in all, or even most, applications. The foremost objectives in the DCN design are summarized below:

1. The supporting network software must be able to run in a wide variety of equipment configurations and hardware options. In particular, it must be

* This research was supported in part by Contract N00014-67-A-0239-0032 from the Office of Naval Research and Grant GK-41602 from the National Science Foundation.

possible to support intelligent terminals such as the GT40 and LSI-11 without requiring major system redesign or loss of critical features.

2. The environment of a process must be uniform throughout the network. Requests for network services such as activation of programs, access to storage media, communication with other processes and interface with the network data base must be standardized so that program and data files can be moved between processing sites with a minimum of reprocessing.
3. The network data base must be accessible to every process, regardless of its location. At least in the case of simple sequential access, the process should not need to be aware of its own location in the network or that of a requested file. The data base cannot be rendered unusable to all users if a single catalog or dictionary fails.
4. Input/output devices such as card readers, line printers and terminals must be accessible from anywhere in the network, either directly or through a spooling facility. An access method must be available for special-purpose resources such as array and transform processors, digital and analog input/output devices, and channels to other machines.
5. A comprehensive capability for performance monitoring and functional control must be built into the network. Using these capabilities it must be possible to experiment with and evaluate a wide spectrum of resource-allocation techniques and methodologies. Systems for data-base access, processor scheduling and process residency should be modular and separable, so that experimental substitutions can be made.
6. Finally, the system should be well-structured and capable of exploitation by graduate students and research workers. Design principles should be widely applicable and consistently applied. To the fullest extent practicable it should be possible to incorporate existing system and application software into the network without modification and to incorporate new hardware and software without extensive system redesign. Reasonable insulation and protection from software bugs should be provided.

Several systems now in existence or under development share some of these objectives. RSEXEC,[9,23] which operates as a subnetwork of the ARPA network, provides capabilities for remote file and device access. As in RSEXEC, the structure of the DCN is based on a virtual-storage and virtual-processor organization. Processes, devices and files can be accessed from anywhere in the network by a standard communication protocol. System integrity is enhanced by a hierarchical collection of loosely-coupled service processes which monitor user requests and poll each other for system status.

DCS,[5-8,22] which includes special-purpose hardware and communications facilities, incorporates a network-wide standard process naming and accessing protocol. As in DCS, DCN processes are named and given a unique identity within the network. Resources are identified and accessed with processes. Critical information is maintained in multiple copies and system damage due to lost system components, such as catalogs, is minimized by use of a standard set of recovery procedures.

Network configuration

The current configuration of the DCN, shown in Figure 1, includes six PDP11 CPU's with a total of about 360K bytes of processor storage and 13M bytes of direct-access storage. Communication bandwidths between the machines vary from 30 bytes/sec to about 300K bytes/sec, depending upon the facilities available. The three PDP11/45 machines, some of which cannot be dedicated to DCN use at all times, constitute the principal supporting resource for the other machines in the network. One of these machines includes special hardware for picture processing, another includes special hardware for real-time signal processing and a third includes interactive and remote-batch connections to the Univac machines. The GT40, PDP11/40 and LSI-11 are used for specialized terminal support and as test beds for the evaluation of small systems. Any of these machines can be connected by asynchronous lines at speeds to 9600 baud.

The 1100 machines provide all program preparation facilities available at present, including the cross assembler, cross compilers and cross link-editor. They are connected to the PDP11 network by either a 1200-baud asynchronous interactive port, by a 9600-baud synchronous remote-job entry port or by both at the same time. This provides facilities for exchanging program and data files between the 1100 machines and the PDP11 network.

SYSTEM ARCHITECTURE

Most DCN operations are performed in the context of a process. A DCN process includes procedure code, data storage and an interface for communicating with other processes and components of the network. Management of this environment is performed by the kernel (see Figure 2), which includes mechanisms for input/output device interface, storage allocation, process scheduling, interprocess communication and other similar functions. A hostel is an abstraction of a virtual environment in which a set of processes can interact with the kernel. An operating system for a hostel includes certain components of the kernel resident in the hostel (the supervisor) and, in addition, certain processes to manage system resources such as input/output devices, catalogs and files.

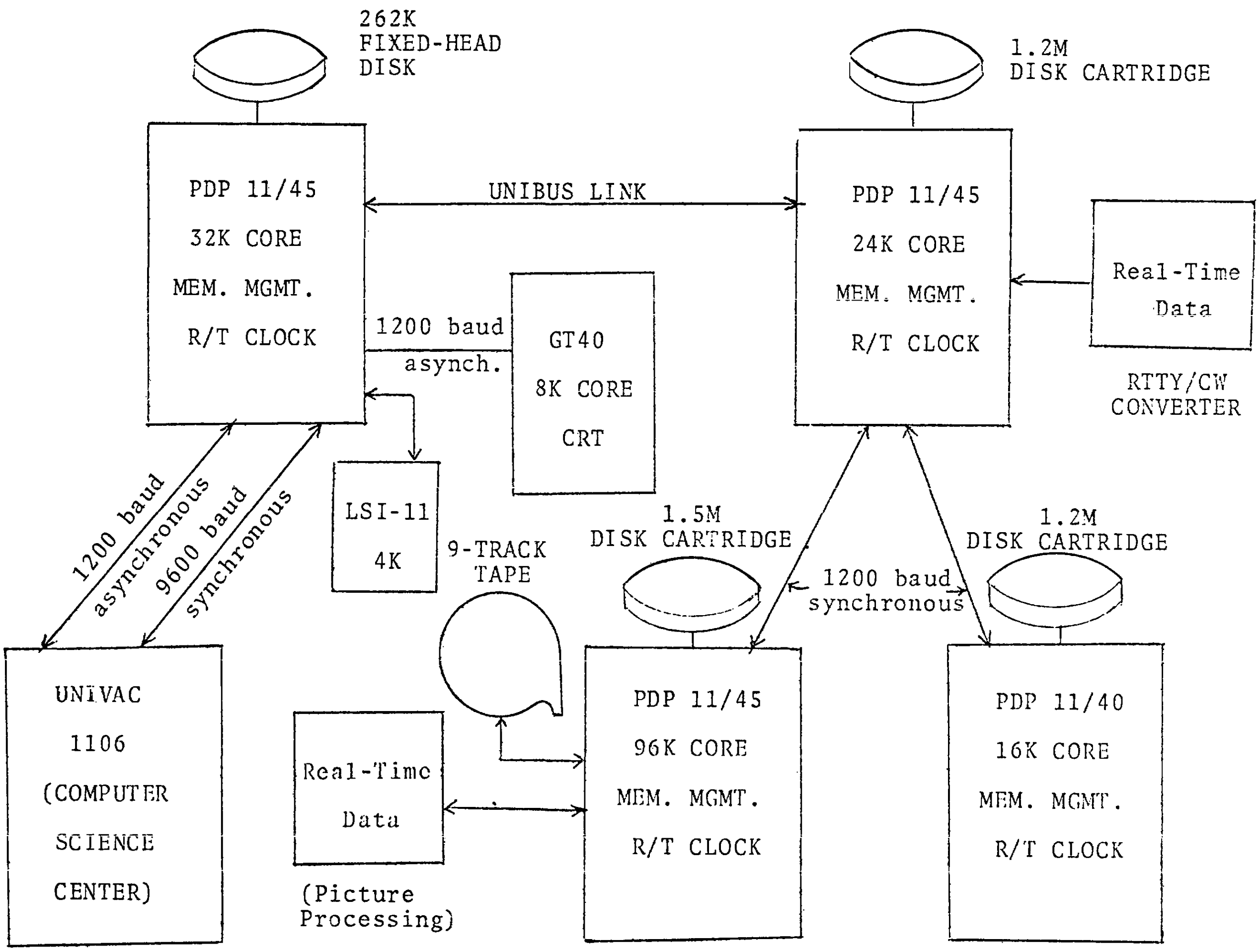


Figure 1—System configuration

There are at present two multiprogrammed operating systems for DCN hostels. One of these, called the Virtual Operating System (VOS), provides full capabilities for user processes in a general-purpose program development and execution system. The other, called the Basic Operating System (BOS), provides a limited subset of these capabilities for special-purpose data collection and processing systems. Both systems can be intermixed in the network and cooperate in providing the services to maintain the distributed environment.

The BOS is designed to operate in a minimum PDP11 configuration including at least 4K words of storage, an operator's console and a communications device for connection to the DCN. The VOS is designed to operate in a PDP11/70 or PDP11/45 configuration including at least 16K of storage, the memory management (hardware relocation) unit, an operator's console and a communications device. A rather extensive set of direct-access storage devices and communication devices can be supported in either system by simply including the appropriate modules at system generation time. Examples of supported devices include fixed and moving head disks, magnetic tapes, synchronous and asynchronous serial communications devices and high-speed interprocessor links.

The VOS and BOS are structured in a similar manner. They consist of the supervisor, a set of user processes and a set of system processes. The supervisor consists of a number of routines to schedule processes, dispatch device interrupts and coordinate interprocess communication. User processes are designed to execute one or more application programs, depending on the particular system configuration. System processes include device processes, which manage the set of input/output devices configured in a particular installation, and service processes, which perform network management functions.

In the VOS, which requires the memory management unit, there are two types of processes: real and virtual. Real processes share a subset of the virtual address space of the supervisor and are designed to

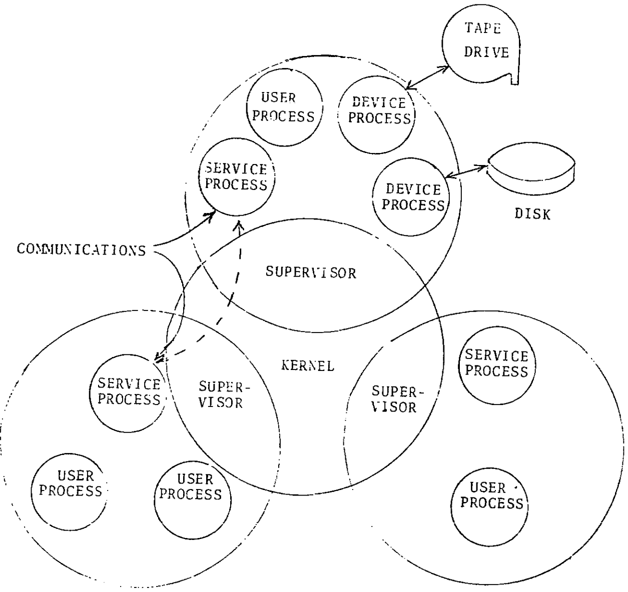


Figure 2—System structure

support peripheral devices, such as the operator's console, communications lines, and direct-access storage devices. Virtual processes have individual virtual address spaces and are designed to support application programs and network management programs. In the BOS all processes are real, including those which support application programs and network management functions.

Figure 3 shows the hierarchical relationships between the various components of the VOS. The BOS is constructed the same way but does not have all of the functions shown. Level zero, consisting of the resident supervisor, provides management of process scheduling, interprocess communication, processor storage and interrupt dispatching. Level one consists of the port dictionary process, which provides access to named interprocess communication channels, or ports. (The port dictionary process itself is accessed by a predefined or well-known-port.) Level two, consisting of a set of device processes, provides an abstraction of the physical devices as a set of processes in which all data transfers are in the form of standard messages. Level three, consisting of a set of swapper processes, manages the virtual-storage operations in the system and supervises segment transfers between processor and direct-access storage. Level four consists of a set of dictionary processes providing access to named virtual files.

The supervisor and levels zero through four are specific to each hostel in the network. Processes at higher levels are free to migrate throughout the network driven by user requests and network management policies. Level five consists of the catalog system, a hierarchical set of processes which provides access to volume and port dictionaries by means of a sequential decomposition of file and port names. Level six, consisting of a set of stream-demon processes, provides sequential access to virtual files. Level seven consists of a set of user processes. Each of these uses the services provided by the preceding levels to create an environment in which a conventional application-oriented program can operate.

Level	Component	Services
0	Supervisor (resident)	message buffering system initialization interval timer process scheduling and timeslicing process synchronization (P, V) processor storage allocation and mapping swap queue management interrupt vectoring
1	Port Dictionary (resident)	port name/address translation
2	Device Processes (resident)	operator's console interhostel communications swap devices
3	Swapper Processes (resident)	demand segment-fetch policy modified LRU segment-replacement policy compaction and garbage collection
4	File System (swapped)	volume dictionaries capability dictionaries access demons
5	Catalog System (swapped)	volume dictionary access port dictionary access capability dictionary access
6	Stream Demon (swapped)	sequential file access
7	User Process (swapped)	message interface file-system interface catalog-system interface interrupt management virtual-storage management

Figure 3. System Hierarchy

Virtual storage management

The PDP11/45 equipped with the memory management unit provides a hardware relocation facility which can be used to realize a virtual storage system. The memory management unit provides three sets of relocation registers, one set corresponding to each of the user, supervisor and kernel states. Each of these states is associated with a distinct processing environment: the kernel state is associated with the basic process scheduling and processor storage mapping functions; the supervisor state is associated with a set of supervisor processes used to manage the input/output and swapping functions; the user state is associated with all other processes, including those for resource management and application programs.

Processor storage mapping functions are handled differently in the three states. In kernel state the mapping is fixed, since all segments are resident in processor storage. In supervisor state the mapping includes the particular segment which contains the process procedure and, in addition, certain procedures of the kernel which can be called by the process. In user state the mapping includes only those segments which have been explicitly requested by the process using a set of supervisor primitives constructed for the purpose.

The virtual storage space available in the three processor states is shown in Figure 4. The segments accessible in kernel state are resident at all times in processor storage. Segments accessible in supervisor state for procedure code are made resident as each process is installed, which can occur dynamically as part of normal system operations. Segments in user space are swapped from direct-access storage to processor storage according to a demand-fetch policy, in which the transfer occurs only upon actual reference. Segments are removed from processor storage according to a modified least-recently-used policy.

All segments in processor storage except those used for message buffers, etc., have images on direct-access storage as segments of files. The remainder are used by the supervisor for scheduling, buffering and message-transmission operations. Associated with every process is a control segment, which contains space for scheduling parameters, general registers, storage-mapping registers, and so forth. As each process is scheduled its control segment is mapped into kernel space, but not into user space. This provides an exceptionally clean and secure interface, since there is no way that a virtual process can access any segment other than those validated by the supervisor.

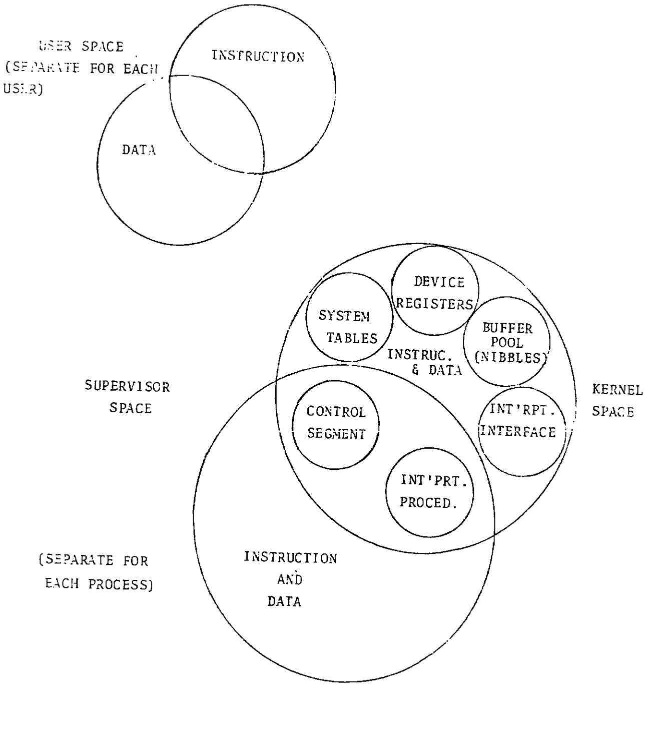


Figure 4—Storage mapping

Interprocess communication and synchronization

Due to the distributed nature of the DCN, processes cannot in general communicate by means of shared storage. Instead, a highly-developed stream-oriented interprocess communication and synchronization facility is provided. This includes the capability for transmitting variable-length messages from any process in the network to any other and provides for the automatic routing of messages between hostels, so that the sending and receiving processes are not in general aware that their correspondent is in the same or a different hostel.

Functional control over message formatting, flow control and routing is provided by the supervisor, together with a set of system processes. The supervisor includes primitives for sending and receiving messages on a byte-by-byte basis. A set of system processes formats them for transmission over a communication circuit or interprocessor link as required. A common buffer pool is maintained in each hostel so that separate buffers for each process are unnecessary.

A message, consisting of a string of bytes preceded by a header, is sent by a process to a named connector called a port. The concept of port, as used here, is similar to those of Walden,[24] Farber[5] and Carr[2] (although called by different names). One or more named ports can be allocated to a particular process on request. A port is always allocated to the receiving process, although any number of processes can send to it. A port name is accessed in the same fashion as a file name. The catalog system (see below) provides access to a port dictionary maintained at the hostel in which the port is allocated. Each entry in this dictionary includes the name of the port, its address and a pointer to the process to which it is allocated.

Ports are identified according to a 16-bit integer called the port address. For efficiency in message transmission and routing, the port address is split into two 8-bit fields: the hostel ID (HID) and the process ID (PID). System processes which transmit messages between hostels use the HID to determine the routing, while the supervisor within each hostel uses the PID to determine the particular port within the hostel.

The use of HID and PID fields in the port address would seem to constrain the mobility of ports. In fact, as shown in Reference 17, recovery from transient system failures requires that all processes with knowledge of a port address be able to correct the address if the process to which it is allocated moves unintentionally. This is a special case of capability revocation (see below). Since in the DCN design intentional process movements are not expected to occur very often, the same correction mechanism is applied in both cases.

THE CATALOG SYSTEM

Files and ports are accessed by a hierarchical set of catalogs in a manner similar to DCS.[6,7] Each catalog is associated with a process which provides the name of another process when given the fully qualified name (see Figure 5). The chain of processes accessed in this way terminates either at the dictionary process for the volume on which the file resides or the port dictionary for the hostel in which the port is allocated. The catalog and dictionary structure is such that each catalog and its access process can be re-created (perhaps on another volume or hostel) if the storage media used to store the catalog or the processor or process which services it fails.[18]

At least one volume at each hostel contains the system catalog, which is at the top of the catalog system hierarchy. This catalog is replicated in each hostel so that failures do not compromise accesses to files in other hostels. Other catalogs may be replicated as well in the interest of efficiency and reliability. A system of continuous consistency checks is now in implementation[17] to insure that the catalog system remains intact in the face of transient system failures.

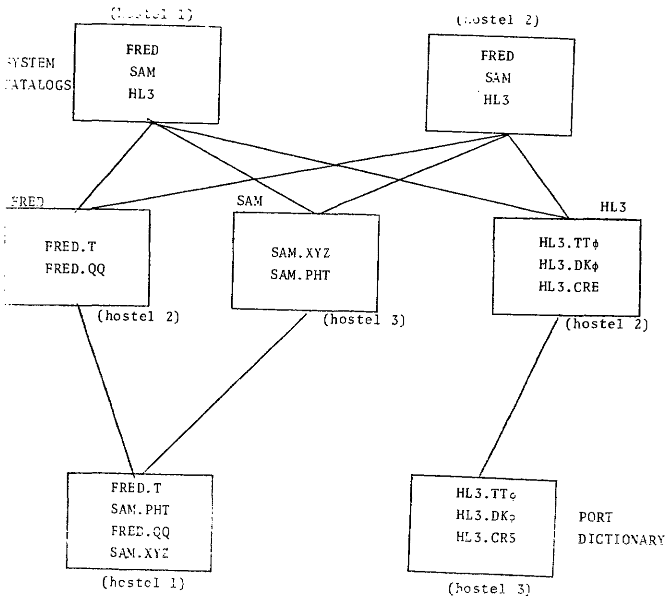


Figure 5—Catalog system

THE VIRTUAL FILE SYSTEM

A file consists of a number of variable-size segments, each of which can be assigned one of several access keys. The segments can be used in any way required—sequentially or randomly—according to the needs of the process. A dictionary is maintained on each direct-access volume in the VOS hostel. Each file on the volume is represented by an entry which includes the fully qualified name of the file and the location, length and access key of each of its segments. The dictionary is maintained by a special process, called the volume dictionary, which responds to requests for additions and deletions of dictionary entries and for accesses to files on the volume.

The segments of a file are accessed by first opening the file and then mapping the specified segment into the virtual storage of the requesting process. A file is opened by sending a message through the catalog system to the appropriate volume dictionary, which returns its port address to the requesting process. A segment is mapped into virtual storage by sending another message to the indicated volume dictionary process, which returns the address of the segment on the volume. When the segment is referenced this address is sent to the swapper, which transmits the segment to processor storage. A service process called the stream demon provides for the connection of a sequentially-structured file to a selected port for input or output. This provides a simple sequential access method for use by system translators and utility programs. More sophisticated access methods are currently being considered.

PROTECTION MECHANISMS AND CAPABILITIES

The environment accessible to a process is completely determined by the set of port addresses and segment addresses known to the process. Each of these are protected quantities and cannot be inspected or changed directly by the process. A new port assignment or segment mapping can occur only as a result of a designated supervisor primitive in which each port address and external address has a local name (or table index) known only to the process.

In order to send a message to a port, a process must first obtain its address from the catalog system and then obtain authorization from the process which is allocated that port. In both of these cases access can be controlled by means of a password or key. In order to access a virtual file segment the file must be opened and the segment mapped into virtual storage. This requires obtaining the port address of the dictionary process, the address of the segment and the segment access key. In addition, every virtual storage segment can be assigned a storage access key. An access generated by an application program is checked against these keys and is permitted only if validated by both of them. Thus, access to a virtual storage segment can be controlled by the file system, using a password required to open the file or change the segment access key, or by the user process, using the same mechanism to change the storage access key.

These mechanisms can be readily generalized and interpreted as capabilities.[3] In a very real sense a port address is a capability which provides access to a port. Only one kind of access permission is implied by this capability—that of sending messages to the port. A segment access key of the appropriate type is a capability which implies permission to read, write or execute the segment. In this context the ability to obtain either capability depends upon the knowledge of the name of the port or file and the password procedures involved.

NETWORK FACILITIES

The mechanisms described so far provide for a uniform treatment of hostels, processes and interprocess communication. It remains to build on this base to show how the operations of the network are organized. It will be most illuminating to consider the appearance of the network from the user's point of view and then describe how his requests are processed by the supervisor and system processes.

The user process

The DCN appears to a user as an hierarchical collection of user processes. A user process can be viewed as a region of processor storage into which can be loaded or mapped one of a number of application programs. In the VOS the region consists of one or more segments of virtual storage as necessary to contain the procedure and data segments of the application program. In the BOS the region consists of permanently resident fixed-size blocks of storage containing shared re-entrant code.

The storage management policy of the user process is characterized by a hierarchical overlay structure. In the VOS most of the 65K-word virtual space is available for the application program. Each overlay consists of a program file, which can contain procedure and data segments. An overlay is initiated by mapping these segments into the virtual storage of a user process and allocating a segment of the process file as a stack. When one overlay calls another, the virtual storage mapping of the calling overlay is saved on the process stack, the virtual storage mapping of the called overlay replaces that of the calling overlay and a new process stack segment is allocated. When the called overlay terminates, the process stack segment and virtual storage mapping of the calling overlay is restored and the stack space released.

Network operations

When the user process is first invoked a simple command language interpreter (CLI) is mapped into storage. The CLI is capable of responding to a number of operator commands, including the following:

1. A set of commands to link to another user process and to return to an old user process.
2. A set of commands to establish an association between a named port and logical port number as used by the application program.
3. A set of commands to send data and control messages to other processes.
4. A set of commands to select an application program (possibly from a library) for execution.

Additional commands are available in some user processes for debugging and auxiliary storage management. A typical set is summarized in Figure 6.

When a user logs on the system he is connected to a system process called the logger. During the log-on procedure a user profile is constructed which contains such information as the default user name to be used in file-access operations and the default hostel name to be used in some port-access operations. Following the log-on procedure the user specifies the name of the user process in which his application program is to run. This is presently done by specifying

Command	Function
ASG <lpn> <portname>	assign a port to a lpn
SEND <lpn> <message> EOF <lpn> <message> ATTN <lpn> <message> CTRL <lpn> <message>	send various types of messages to processes identified by lpn
DISPLAY <bgnadr> <nwords> ALTER <bgnadr> <value> . . . SENSE <lpn> TEST <nlines> <code> RESET	commands used for debugging and system testing
FORK <lpn> <message> JOIN <message> END <message>	commands to initiate and terminate user processes and to control synchronization between calling and called processes
LOAD <lpn> START <adr> <parameters>	commands used to load application programs and initiate execution
LOGON <parameters> PROFILE <parameters> LOGOFF	commands to log on and off and establish user profile (defaults)
RUN <portname> <parameters>	combination of ASG, LOAD and START
LINK <portname> <message>	combination of ASG, CTRL and FORK

(<lpn> is logical port number)

Figure 6—Command Summary

a hostel name (possibly remote) together with a user process name in that hostel.

Connections to the specified user process are established by searching the catalog system for the process name. If found the user is connected immediately to that process; if not, a new user process is created in the hostel indicated in the user profile and the user connected to it. In either case, the user profile is transmitted to the new process, so that it has access to all the files and ports of the old one. In a similar manner a user process can initiate another user process to perform some function while the old process either continues or is suspended until the new process terminates. When communications are terminated between the user and the original user process, the logger once again assumes control and the user can either log off or select a new user process.

APPLICATIONS SUPPORT CURRENTLY AVAILABLE IN THE DCN

The development of the DCN has reached the point where many application programs can be supported in the network environment. In a complex system such as this, confidence in the utility and capability of the system is gained only through user experience. The facilities described below were developed primarily to test the suitability of the DCN for their operation.

A special emulator, now in development, provides a virtual-processor environment in which programs written for the manufacturer's Disk Operating System (DOS) can execute. The emulator includes a set of supervisor-call interpreters compatible with DOS and with which commands can be entered into input/output devices and sequential files. Only those programs which are "well behaved" in the DOS sense can be supported (i.e., do not reference device registers directly). Although load modules produced by the DOS assembler, FORTRAN compiler and linker are well behaved in this sense, some of the system components within DOS, in particular the FORTRAN compiler itself, require direct file access, which is not supported in the DCN. In the FORTRAN compiler direct file access is required only to accomplish an overlay function, which is provided in a different fashion by the VOS itself. Work is under way at the moment both to provide direct file access and to change FORTRAN to call the VOS overlay facilities rather than its own.

Several cross assemblers and cross compilers are available on the 1100 machines, including the assembler, link editor, and the SIMPL and ULP compilers.[1,13,19] Programs in these languages are usually compiled and link-edited on the 1100 machines and then transmitted to the PDP11 network by means of real-time or remote-batch facilities. In addition, a PDP11 version of the ULP compiler and real-time interpreters for LISP[10] and BASIC are available on the PDP11 network.

Several application packages have been constructed for use in the DCN. These include two and three dimensional graphics transformations (which can be used in real-time by the GT40),[20] an interactive instruction-trace monitor and paging simulator, a VOTRAX speech synthesizer driver and various signal processing and recording routines for satellite tracking and telemetry.

SUMMARY AND CONCLUSIONS

The facilities of the DCN are now being used to develop several applications packages, including interactive graphics and image-processing systems, real-time data collection and processing systems and interactive calculator systems. These tools are also being used to study program behavior in the distributed environment. The distributed nature of the network and the organization of the VOS and BOS operating systems have greatly facilitated these tasks and have aided in the refinement of the DCN concepts.

At present many of the original design goals for the DCN have been realized. Still remaining to be developed are additional mechanisms to aid in a strategy for automatic file and process migration, load leveling and performance monitoring. These mechanisms are of course at the heart of the concept of the distributed network and indicate the direction of future research. The most important aspect of the DCN at its present level of development is that it forms the basis of a set of tools to study operating systems architecture, data-base organization and communications strategies as applied to networks supporting distributed resources.

REFERENCES

1. Basili, V. R., *The SIMPL Family of Programming Languages and Compilers*, Computer Science Technical Report TR-305, University of Maryland, June 1974.
2. Carr, C. S., S. D. Crocker, and V. G. Cerf, "HOST-HOST Communication Protocol in the ARPA Network," *Proc. AFIPS*, 1970 SJCC, pp. 589-597.
3. Dennis, J. B. and E. C. van Horn, "Programming Semantics for Multiprogrammed Computations," *Comm. ACM* 9, 3, March 1966, pp. 143-155.
4. Digital Equipment Corporation, *Introduction to RSX11-D*, Form DEC-11-OXINA-A-D, Digital Equipment Corporation, 1974.
5. Farber, D. J. and K. Larson, "The Structure of a Distributed Computer System—Communications," *Proc. Symposium on Computer Communications Network and Teletraffic*, Brooklyn Polytechnic, 1972.
6. Farber, D. J. and K. Larson, "The Structure of a Distributed Computer System—Software," *Proc. Symposium on Computer Communications and Teletraffic*, Brooklyn Polytechnic, 1972.
7. Farber, D. J. and F. R. Heinrich, "The Structure of a Distributed Computer System—The Distributed File System," *Proc. ICC—Impacts and Implications*, October 1972.
8. Farber, D. J., "Data Ring Oriented Computer Networks,"

in *Computer Networks*, R. Rustin (Ed.), Prentice Hall, Inc., Englewood Cliffs, N.J., 1972.

9. Kahn, R. E., "Resource-Sharing Computer Networks," Proc. IEEE 60, November 1972, pp. 1397-1407.
10. Kirby, R. L., *PDP11 LISP Documentation*, Computer Science Technical Report TR-400, University of Maryland, August 1975.
11. Lay, W. M., D. L. Mills and M. V. Zelkowitz, "Design of a Distributed Computer Network for Resource Sharing," *AIAA Computer Network Systems Conference*, Huntsville, Alabama, April 1973.
12. Lay, W. M., D. L. Mills and M. V. Zelkowitz, "Operating Systems Architecture for a Distributed Computer Network," *ACM Conference on Trends and Applications of Minicomputer Networks*, Gaithersburg, Maryland, April 1974.
13. Lay, W. M., A. K. Stebbens and J. A. Pollizzi, *PDP11/ Univac 1108 Cross Assembler System*, Computer Science Technical Report TR-422, University of Maryland, October 1975.
14. Mills, D. L., *An Overview of the Distributed Computer Network*. Computer Science Technical Report TR-413, University of Maryland, October 1975.
15. Mills, D. L., W. M. Lay and J. Pollizzi, *The Virtual Operating System for the Distributed Computer Network*, Computer Science Technical Report (in preparation).
16. Mills, D. L., *The Basic Operating System for the Distributed Computer Network*, Computer Science Technical Report TR-416, University of Maryland, January 1976.
17. Mills, D. L., *Transient Fault Recovery in the Distributed Computer Network*. Computer Science Technical Report TR-414, University of Maryland, January 1976. Condensed version in: *Proc. NBS IEEE Trends and Applications 1976 Symposium* (to appear).
18. Mills, D. L., *Dynamic File Access in a Distributed Computer Network*, Computer Science Technical Report TR-415, University of Maryland, January 1976.
19. Mills, D. L., *Structured Programming and Compiling in a Minicomputer Environment*, Computer Science Technical Report TR-339, University of Maryland, October 1974. Condensed version in: *Proc. IFIP TC-2 Working Conference on Software for Minicomputers*, Lake Balaton, Hungary, September 1975.
20. Mohr, J., *A Graphics Supervisor for the GT40*, Computer Science Technical Report TR-440, November 1975.
21. Ritchie, D. M. and K. Thompson, "The UNIX Time-Sharing System," *Comm. ACM* 17, 7, July 1974, pp. 365-375.
22. Rowe, L. A., M. D. Hopwood and D. J. Farber, "Fail-Soft Behavior in the Distributed Computer System," *IEEE Symposium on Computer System Reliability*, New York, April 1973, pp. 7-11.
23. Thomas, R. H., "A Resource Sharing Executive for the ARPANET," *Proc. AFIPS*, 1973 NCC, pp. 151-163.
24. Walden, D. C., "A System for Interprocess Communication in a Resouce Sharing Computer Network, *Comm. ACM*, 15, 4, April 1972, pp. 221-230.
25. Zelkowitz, M. V., "Simulation and Implementation of Computer Networks," *Proc. Thirteenth Annual ACM Washington Chapter Symposium*, June 1974.

A network-oriented multiprocessor front-end handling many hosts and hundreds of terminals

by W. F. MANN, S. M. ORNSTEIN and M. F. KRALEY
Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

ABSTRACT

The authors discuss the design of a large-scale front-end computer in terms of system requirements, available technology, and the authors' experience with the ARPANET. The design is contrasted with that of the ARPANET TIP. Issues discussed include the choice of hardware configuration (CPU requirements, reliability, modularity, terminal interface units), what facilities to provide in the front-end, the communications protocol between front-end and Host, flow control on various inter-computer data paths, and data buffering strategies. The resulting system is being installed at the Research Computer Center at Bolt Beranek and Newman Inc.

INTRODUCTION

In recent years there has been a tremendous increase in the use of computer terminals and other types of remote access facilities. Part of this increase can be attributed to the proliferation of time-sharing systems, which by their nature require multiple terminal access. But more generally, the growth of data communications has gone hand-in-hand with increasing remote usage and with the interconnection of remote facilities to form various kinds of computer networks. Servicing access ports, since it can consume a significant fraction of the main computer's power, has been off-loaded both to specialized I/O channel processors and to more general "front-end" processors. As time has passed, and because of the somewhat specialized nature of the job, the design of such machines has evolved into an independent art form.

Our group at Bolt Beranek and Newman has been deeply concerned with the evolution of this sort of machine through our work in the development of the ARPANET.[1] Designing the Interface Message Processors (IMPs)[2] and Terminal Interface Message Processors (TIPs),[3] both based on the Honeywell 316 line of computers, and later the Pluribus IMP,[4] has involved us in many issues of communications processing. Recently we have built yet another front-end computer to provide connections among several on-site Host computers, a number of shared peripheral control computers, several network ports, and a large (and increasing) number of terminals of various types. As such requirements are becoming increasingly common, we felt it would be useful to discuss some of the design issues we have confronted and the particular solutions we have chosen.

Because requirements have varied widely from one site to another, many front-end processors have developed as *ad hoc* solutions to particular problems (e.g., Reference 5). It has often proved difficult to estimate how much computer power will be required and to anticipate how fast requirements will increase, and in a number of instances the resulting systems have not proved terribly satisfactory.[6] In some cases the front-end processor was underdesigned for the known requirements and eventually ran out of processing bandwidth, memory address space, I/O ports, or memory channels; in other words, the system reached some sort of growth limit. In other cases one or more of these limits was reached because the requirements expanded much more rapidly than expected.

One solution to these difficulties is a system capable of relatively easy expansion. Over the past several years we have been developing just such a line of machines, known as Pluribus, and have reported on this work at earlier National Computer Conferences.[4,7] The Pluribus is a modular minicomputer-multiprocessor which cannot only meet expanding requirements through a smooth process of growth, but can also provide extreme reliability when used with an adaptive program able to utilize redundant hardware resources. In the following paragraphs we provide a brief update to our earlier reports.

A number of Pluribus machines have been built and are now working in a variety of environments. One is installed in the ARPANET as a high-speed IMP; it is currently connected to four Hosts as well as to five network lines, a configuration larger than could be handled by previous IMPs.[4,7] A second machine, of approximately the same size but with different I/O equipment and with high reliability requirements, has been installed as a Host to provide pre-processing of seismic data collected through the network.* Two Satellite IMPs (with special time-keeping interfaces and a program which implements a broadcast capability) are being used in a development program,[8,9] and several small, single-processor, Private Line Interface (PLI) machines[10] have been built which can provide an ARPANET user with (secure) communication over what appears to him to be a point-to-point private line. Finally, a large 13-processor version of an IMP is undergoing testing and evaluation.[11]

In general, our ideas concerning modularity and reliability appear to be holding up very well, and this has encouraged us to consider further applications. We have proposed using Pluribus machines in a number of other computer networks now being considered, and we are presently engaged in a study to consider the suitability of using a large version of the machine as the basis for a person-to-person message-handling system. In addition we are beginning to explore the extension of the basic architectural notions to the next generation of multiprocessors.

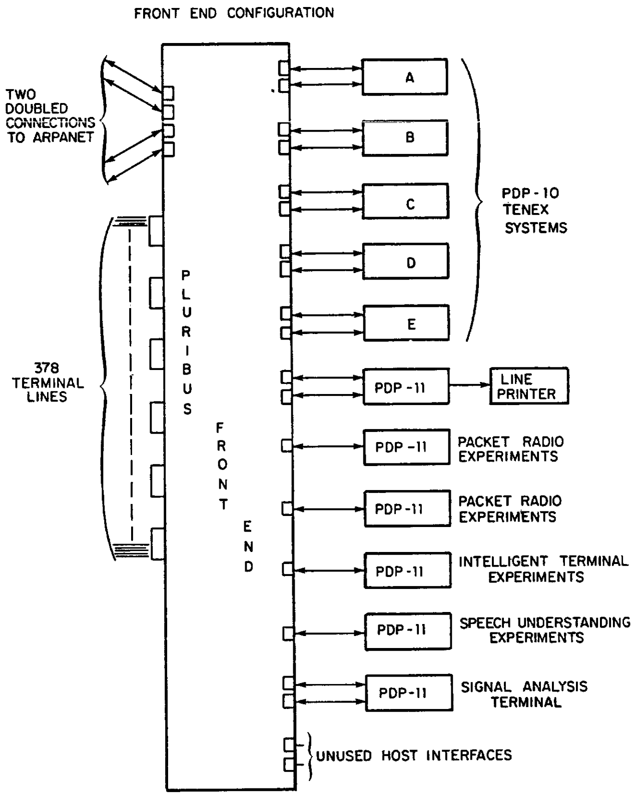


Figure 1

FRONT-END REQUIREMENTS

The growth of the ARPANET has created increased demands on a number of network sites which provide resources useful to many people. Our own Research Computer Center[12] is an example of such a resource and both the central facility and its accessing mechanisms have rapidly been reaching saturation. The central (PDP-10) TENEX facility has grown by replication so that several TENEX systems are now housed in a common center; this only intensifies the accessing and inter-communication problems, however, since it is desirable for all terminals to be able to attach to any of the Host systems, for the Hosts to be able to communicate with one another, and ideally for the Hosts to share access to I/O devices such as high speed printers. Figure 1 shows the configuration we have chosen to provide the required interconnections.

Terminals are currently connected to our TENEX time-sharing systems in one of two basic ways: either via a more-or-less traditional route using a terminal scanner connected to a PDP-10 I/O bus (through a characteristically messy patch panel), or via network access. Terminals that come in through the network either connect to a local Terminal IMP (TIP) or are associated with some remote network site. We wanted to find a solution which provided a homogeneous method of access to multiple Hosts for all local terminals, i.e., all those that previously connected through either the patch panel or a local TIP. Manual patching of terminals was recognized as a severe operational problem. The number of terminals being serviced was already in the hundreds, and may eventually grow to as many as a thousand. Furthermore the number of Host computers at the site was being enlarged to handle increasing demands and it was clear that we should be able to attach at least fifteen or twenty Hosts of various sorts. Finally, if a single element was to be used to connect everything together, it had to be extremely reliable since all communications would pass through it. A list of these communications includes the following:

1. All intra-site traffic between local TENEX systems.
2. All traffic between local TENEX systems and remote sites in the network.
3. All traffic between local TENEX systems and various local specialized Hosts (e.g., a PDP-11 which runs a shared line printer facility for the TENEXs.)
4. All terminal traffic (both local and remote) to all of the TENEX systems.
5. All local terminal traffic to access remote Hosts.

* The seismic Communications and Control Processor (CCP) is a four-processor six-bus Pluribus system that serves as the central data switching and control node in the ARPA-sponsored seismic data collection network. The CCP is installed at the Seismic Data Analysis Center in Alexandria, Virginia.

MACHINE SELECTION

In keeping with our desire to off-load user accessing tasks from the Hosts, we first considered connecting all local terminals through regular 316 TIPs. We needed to handle over 350 terminals to begin with and since a TIP can handle at most 63 lines in terms of hardware, buffering, and processor capacity, this solution would have required six TIPs. That seemed very expensive, not particularly reliable, and would have required many interconnections among the TIPs. Furthermore the 316 TIP suffers from being limited to fixed device-buffers and to a set of rigid protocols, both TIP-to-user and Host-to-Host, and also suffers from lack of adequate flow control between TIP and user.[13] Its total bandwidth is limited by the 316's CPU and memory channel hardware, and by the TIP-IMP interface software. Most of these limitations are directly due to the amount of memory which can be addressed by the 316. This memory limitation cramps the current TIP software: new features are difficult to add, device buffers are inadequate, and major improvements are extremely difficult and time-consuming. Finally, the TIP protocols for dealing with terminals via network connections were designed with network-wide generality in mind and as such are costly in terms of Host bandwidth.[14] We felt that something much simpler would be more suitable for local terminals. For all of these reasons we decided to look for a better solution.

Although there was some temptation to make use of front-end technology already developed for such machines as the PDP-11,[5,6] it was clear that the mass of the problem (number of lines, quantity of traffic, etc.) precluded the use of a single such machine. At best, a collection of such machines would be required, with many interconnections; and since we had already developed such a multiple machine, we next turned our attention to the Pluribus as a possible solution. Perhaps not surprisingly, we found that the Pluribus could meet all of our requirements. It is modular and can be expanded in a variety of domains: more memory can be added as required for additional programs, buffers, etc.; more processors can be added to increase processing bandwidth without reprogramming; a large address space, combined with a modular physical structure, power, and cooling permits enormous expansion of I/O facilities.

This meant that we could start with a reasonable set of hardware and expand as requirements increased. In addition the Pluribus had already been programmed as an IMP and could handle all required Host connections with plenty of room for expansion. Finally, a great deal of attention had been given to matters of reliability.[7] We had accepted the idea that parts of the machine would gradually and inevitably break, and had striven to produce a machine that was totally fail-soft and which could be repaired while it continued to work. This unique capability is extremely desirable in such a central element of the computer center.

The Pluribus thus offered an attractive point of departure for a solution. A number of ingredients were missing, however. In particular, none of the TIP features (hardware or software) existed. For the hardware we decided that we could adapt our own 316 Multi-Line Controller[15] to the Pluribus and use as many as were needed to service the desired number of lines, a choice discussed at some length below. The absence of TIP software was not such a serious drawback as it might appear since we would have wanted to rework the algorithms in the old TIP anyway to improve their efficiency and capacity. We therefore decided to proceed with a design based on the Pluribus.

DESIGN ISSUES

Dedicated function machines

One of the issues discussed at the time of the 316 TIP design was whether to provide terminal handling and servicing functions in the same machine that housed the store/forward IMP functions, or whether to perform these functions in a separate machine that would deal with the IMP as a regular Host does. We opted at that time for the single machine solution, arguing that the efficiencies of intra-machine communication would reduce overhead to the point that a single machine could do the job. Debate has nonetheless continued, a strong counter-argument being that the IMP code is relatively unchanging and that the perpetual modifying and tuning required of the user-interfaced TIP code can endanger the more central IMP functions. Bugs in new TIP releases can more easily affect the IMP program if both programs reside in the same machine.

This argument, though still true to a certain extent, was outweighed by the cost savings and enhanced reliability provided by a single large Pluribus machine. Under the Pluribus philosophy all functions are combined into a single carefully partitioned program and all processors work separately at whatever needs to be done. This approach facilitates load sharing under shifting workloads and *enhances* reliability. In the event of a processor failure, the other processors continue doing all necessary jobs. If a memory unit fails, another memory unit is pressed into service until the first unit can be repaired. In contrast to the usual state of affairs, we expect larger Pluribus machines to be more reliable than smaller ones, since more resources are available to be redistributed in case of trouble. The Pluribus approach to reliability is based on redundancy and self-checking. Bugs of all sorts are anticipated, including lingering software and hardware bugs, and mechanisms are included both for detecting trouble and for correcting the symptoms.

Terminal interface hardware

Another design issue was selection of the hardware to provide the actual terminal connections. When we began, we assumed that the technology had improved since 1970, when we had been forced to design our own Multi-Line Controller. We thought that a convenient off-the-shelf solution would be available and in particular we were hoping to find a multiplexor or concentrator whose high speed output we could bring directly (without demultiplexing) into our machine. We wanted to do demultiplexing within the machine, through a combination of hardware and software, in order to eliminate the need for a large number of individual line interfaces. Using a multiplexor would also permit us, later on, to install multiplexors at remote locations, near remote terminal clusters, thereby saving on line costs.

Unfortunately we had to deal with at least three terminal types: Teletypes, TI Silent 700s, and 1200-baud scope terminals. Furthermore we had to be able to connect any of the three types of terminals to any system port, and this meant dynamic programmable configuring of the multiplexor. We found that the flexibility in commercially available multiplexors was, for the most part, limited to strapping options. While "Autobaud" features are available, they do not reduce the part of the high speed line bandwidth used by the port's subchannel. Thus with Autobaud we would have had to allocate 1200 baud to all ports even though typically a third of the terminals in use at any one time are 110-baud Teletypes. If we took the multiplexor high speed line directly into the Pluribus, it would have meant either more sophisticated hardware demultiplexing, or a large amount of wasted processor and memory bandwidth.

Thus our review of the multiplexor-concentrator field was disappointing. More and more intelligence is being added to these systems but they are still being treated as subsystems separate from the computers which must eventually service the lines. From our point of view, inadequate industry attention is being given to allowing demultiplexing to be carried out directly within the recipient computer system and allowing it to control the multiplexors themselves.

The 316 TIP's Multi-Line Controller is, in fact, a local concentrator. It is completely programmable as to individual port line characteristics and reports character/line number pairs to the TIP. It is designed to interface directly to a computer, fetching characters from and storing characters into memory, and has all the necessary features to permit program control of line characteristics. In short, the MLC seemed eminently suitable for use with our local terminals and datasets, so we adapted its 316 interface to attach it to the Pluribus.

The communications protocol

A third major design question was how to control the terminals. In the 316 TIP, a quite general set of protocols was provided whereby a terminal user could carry on a single control dialogue directly with a Host-like program in the TIP. By talking to this "mini-Host" a user could specify information about what network Host he wished to be connected to, what transmission modes he wanted, etc. Furthermore the "mini-Host", by implementing a modest version of the general network-wide Host-to-Host protocol,[16] in particular the terminal-handling function (known as the TELNET protocol),[16] permitted a user to communicate directly with any Host in the network that also implemented the TELNET protocol. Unfortunately, server Hosts have found the standard Network Control Program (NCP)[16] to be expensive in terms of machine cycles, and the harmful effects of this clearly increase with the number of terminals in operation. We therefore decided on a simplified form of communication between our terminal-handling code and the TENEX Hosts. At the Host-to-Host level we use a special protocol which provides only for buffering and passing characters between the terminals and the Hosts. It does not include the capacity to recognize escape characters in the data stream and thus deflect some of the stream to its own intelligent mini-Host system; instead, the character stream is treated transparently.

The user's interface to local Hosts

The decision to put all of the character interpretation functions in the Host raises the issue of how the terminal-Host connection is established. Since escape characters are not provided, a user cannot specify, in a subset of his character stream, such things as where he wishes to be logically connected. Instead, when he starts to use his terminal, he is automatically connected to some local Host. Some terminals are "soft-wired," via the front-end, to particular Hosts. The other terminals are defined as "wild," which means they are automatically connected to any suitable local Host. Furthermore, if the Host to which a particular terminal is normally soft-wired is inoperative when the user starts to use his terminal, a suitable alternate can automatically be chosen. The point is that initial logical connection is made to a particular real Host, not a mini-Host in the TIP. When a user first starts typing he is talking to that Host just as with a more traditional direct connection. This avoids the sort of double login (first to the TIP and then to the Host of choice) that we have experimented with (somewhat unhappily) in the ARPANET.[17] It furthermore removes the more sophisticated (space consuming but not bandwidth consuming) functions from the front-end to the larger Host which is better suited to handle them.

The user then talks directly to a command interpretation system in the Host. If he needs to switch his logical connection to a different Host, he uses the specific Host commands defined for such purposes. If the switch is to a different *local* Host, then a command is returned from this Host to the front-end instructing it to switch the user's logical connection to the new Host. Thereafter his character stream passes through the front-end directly to the Host of interest without occupying the initial-connection Host at all. All of this is in the spirit of Reference 18.

Interfacing to remote sites

The approach described above is intended to satisfy the needs of a local user on one of our local Hosts. Users of the Pluribus front-end who desire to access remote network Hosts will initially have to pass their character streams through TELNET on a local Host (since the front-end has no such facility). This is a burden on the local Host which the 316 TIP does not impose.

To relieve the local Hosts of this burden, we intend to add the NCP and TELNET functions to the Pluribus front-end. This in no way interferes with the simplified protocols described above for local users of local facilities. The front-end software currently consists of four logical modules: an IMP module, a terminal handler, a front-end protocol handler, and a Host/IMP protocol handler (see Figure 2). Together these provide the usual IMP functions and, in addition, act as a very large terminal scanner for the collective TENEX systems. With the addition of a fifth, NCP-TELNET, module, the front-end could connect its terminals directly to remote Hosts, eliminating the need for a user to log onto TENEX merely for the purpose of running TELNET in order to gain access to a foreign system.

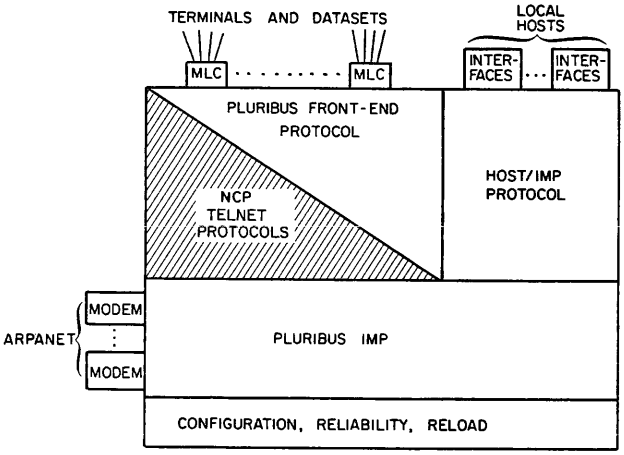


Figure 2

More generally, the advent of an enlargeable front-end machine opens up the possibility of off-loading still more of the communication-handling function from the Host computer. Once we have the NCP-TELNET module we plan to use it to accept incoming TELNET connections and serve as an interface between *remote* users and our TENEX Hosts. The remote user would establish a TELNET connection to the Pluribus front-end, and would from then on appear to be a local terminal to the TENEX Hosts. This kind of facility is the main thrust of several other ARPANET front-end projects.

Flow control—Host to IMP

Flow control is an issue in the design of all communications systems. One way to stop unwanted flow of traffic over an interface is simply to block it until such time as we want it to resume. This expedient has the advantage that it allows a very simple set of rules for the users of the interface. Its major drawback, however, is that a single electrical "pipe" is treated as a single logical path. In fact, some types of connection, such as the one between a Host and an IMP, provide for many concurrent logical paths. While blocking one or more of them from time to time may be necessary, having only one blocking mechanism for the entire pipe is clearly undesirable.

In particular, if we consider the connection as providing only two paths, one for terminal-Host access and one for other Host-network traffic, then we can see that while it may become necessary to block network traffic (due, let us say, to momentary lack of some network resource such as buffer space or traffic control blocks), it is certainly not desirable to let this interfere with terminal traffic which does not require the temporarily depleted resource at all.

We discussed the possibility of using two separate hardware interfaces between the Pluribus front-end and the TENEX Hosts, one for regular network messages (including intra-site Host-to-Host traffic) and the other for front-end traffic. This was motivated by blocking problems observed at the 316 TIP-Host interface. This issue is somewhat similar to the one-or-two-machines question and here again we opted for the shared path solution, modifying the software to provide an extended, non-blocking IMP-Host protocol. The essential idea of this protocol is that the receiver will always remove his messages from the pipe, thereby guaranteeing that the pipe will keep flowing. If for any reason the receiver is unable to process a message, he merely throws it away. This of course requires an acknowledgment/retransmission protocol to properly account for messages. This protocol is being implemented as a new ARPANET standard.[19]

Flow control and buffering—Front-end to TENEX

Flow control is also an issue at the terminal level. As a terminal user types, characters flow to the Host and eventually into some buffer in the Host memory which is associated with that particular user. The program in the Host takes characters out of the buffer, inspects them, and processes them. The buffer is required because the program does not check each line constantly but only in turn as determined by the swapping algorithm. The larger the buffer, the more leisure is permitted to the program, but the greater is the cost in space (and response time).

A flow control mechanism is required to guarantee that the flow of characters will stop should the buffer ever fill up. In TELNET the mechanism works as follows. Initially the receiver allocates some amount of buffer space to the line and informs the sender of this space allotment. Each time the sender sends off characters to the receiver, it reduces the space it believes is still available at the receiver by an equivalent amount. Correspondingly, as the receiver removes characters from the buffer it informs the sender of the change, and this increases the receiver's available space count accordingly. Since only changes are reported, precautions are required to assure that the sender's and the receiver's notions of available space do not become permanently separated through some communications error.

In addition to this buffering in the receiver (the Host for input and the TIP for output) there is additional buffering in the sender. For input, this allows the user to continue typing even when his characters cannot be sent off immediately because of lack of space in the Host. In the 316 TIP fixed buffering is used, and the amount of space available for this sender buffering is small. So, if the remote Host is at all sluggish in taking the characters, the buffers quickly fill and the user receives an alarm indicating that some of his typing has been thrown away. Even worse, by the time he actually stops typing, the blockage will generally have cleared up and the last few typed characters may be accepted, leaving a gap where the momentary blockage occurred. The user must then delete back to the missed section and resume from there, an awkward procedure at best.

The Pluribus's ability to incorporate a large memory and a large number of terminals makes a buffer pool arrangement more sensible. In the Pluribus front-end, if space has been allocated for the user in the receiving Host, the user's characters go directly into a message for the Host. If the space allocated in the Host becomes momentarily used up (because the Host does not empty its buffer rapidly enough) then sender buffers are borrowed as required from the buffer pool in the Pluribus to hold any input typing until the Host empties the "pipe" at its far end. The buffer pool is large and is shared not only among all users for such backup of input buffering but also, as we will see below, for buffering of output.

With this scheme, given a sizable buffer pool, the user should never see any blockage. Occasionally a user will see a delay in remote echoing (from the Host) as the Pluribus stores up input while waiting for the Host. A person's natural tendency not to type on blindly until he sees echoing will restrain him from typing too far ahead and thus depleting the pool. For controlling paper tape readers, etc., we plan to provide flow control of the XON/XOFF type.

Flow control and buffering—TENEX to front-end

If we now consider output from the Host to the terminal we see a somewhat different problem. Output typically consists either of slow character-by-character echoing of user input or of substantial bursts of output characters. The output burst rate is typically many times the input rate. Terminals vary in the speed at which they can accept output and some (such as line printers) may periodically refuse characters for some time (e.g., during a page eject). In the TELNET protocol, output flow control is handled exactly the same way as input flow control, through the use of allocated space in the receiver (TIP). The buffers used for output are typically larger but the mechanism is the same. Buffer sizes vary depending upon the type and speed of terminal and are fixed when the terminals are installed.

In the Pluribus front-end a different approach is used which takes advantage of increased buffering capability to simplify procedures and to virtually eliminate sender buffering requirements in the Host. The Host simply sends output *ad libitum* to the Pluribus. The Pluribus allocates (50-character) buffers from its pool, as required to hold the output that comes in, until it can be sent to the terminal. If the rate of Host output exceeds a terminal's rate for a period, the amount of buffering required will eventually rise above some pre-defined clip level for that terminal. At that point a "stop sending" message is sent to the Host. Output for the terminal will continue to be accepted and may absorb further buffers from the pool. When the Host heeds the request to halt output this "overshoot" process stops. As the terminal catches up, buffers are released back into the pool and when the queue of output buffers drops below a second, lower clip level, a "resume sending" message is sent to the Host. This sort of "servo" operation allows the front-end to maintain terminal output flow over a wide range of Host behaviors. It also means that the Host sees only occasional "wait-a-minute" and "O.K.-to-resume" messages instead of an allocate message for each small set of output characters.*

* One might ask, why not make changes similar to all of these in the 316 TIP? The answer is threefold. First, TELNET is a network-wide standard and changing it would require corre-

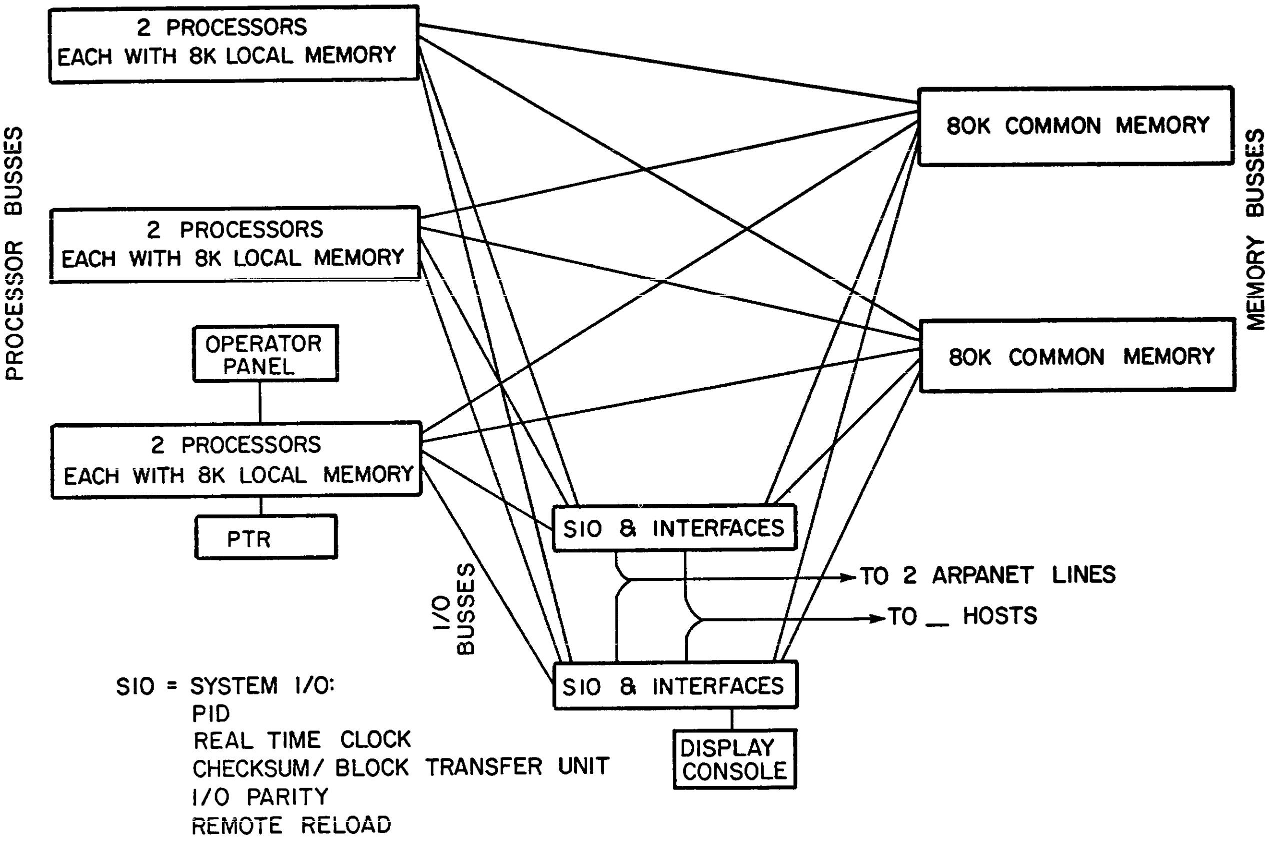


Figure 3

HARDWARE CONFIGURATION

The Pluribus front-end physically occupies eight racks, six for the central machine and two to house individual line interface units (see Figure 3). Six BBN Multi-Line Controllers (MLCs) are included in the initial system, providing interfaces for up to 378 terminal lines. The Host, high-speed network lines, and MLC interfaces are mostly duplicated, on separate I/O busses, for increased reliability. Our bandwidth calculations indicated that four processors were more than sufficient to handle the load of the entire system; reliability considerations led us to include three processor busses, each with two processors. The configuration includes enough memory (80K words on each of two memory busses) to allow for ample buffering, both for store-and-forward and for terminal traffic. There is also room to expand the code to provide many features, such as the improved user interface to TELNET, long desired in the 316 TIP but unimplemented due to lack of space.

The system design takes full advantage of the reliability features of the Pluribus. Sufficient resources of each type are provided so that, in the event of any single failure (hardware or software), the system will continue to run without objectionable degradation, and without human intervention. In case of a hardware fault, the offending component may be diagnosed, debugged, removed, or replaced while the remainder of the system continues to run. The system will automatically incorporate newly fixed components back info the system.

We did not provide any redundancy in the case of the actual terminal concentrator, the MLC. We felt justified in this because of the extremely low failure rates of the MLC in our network experience. Since there are six MLCs in the configuration, a total MLC failure would only knock out a sixth of the TIP's capacity. The I/O bus to MLC interfaces are duplicated, so failures within the Pluribus, or maintenance activities, will not knock out MLCs.

Since this machine is an IMP, it has all of the usual network remote debugging aids such as DDT, reload, verification, and restart facilities. A local display terminal, programmer's console, and paper tape reader augment these capabilities.

sponding changes everywhere else in the Network. Second, our protocol does not address all the issues involved around the Network. Third, the techniques we use would require more main memory than the 316 TIP can address.

THE FUTURE

As discussed above we plan to implement NCP and TELNET functions in the Pluribus front-end in order to off-load these tasks from our TENEX Hosts. Plans for the future also include the development of a flow control procedure for externally clocked terminals, and support for various exotic terminals. While the MLC will handle most if not all asynchronous terminals, we see a pressing need to handle devices such as IBM 2780s, 2260s, and 3270s, which communicate in assorted synchronous line protocols. The Pluribus hardware already includes interface cards which can handle synchronous lines, and an earlier project in which an RJE terminal was tied to an IMP through a Pluribus-based mini-Host has demonstrated the operation of a 2780 using this interface.[20] We see no particular problems in incorporating this approach directly into our front-end eventually, although there are no immediate plans to do so.

ACKNOWLEDGMENTS

A number of people have contributed to the work described herein. Front-end requirements were largely motivated by discussions with Robert Clements (who also made the required modifications to TENEX), Stephen Chipman, and Theodore Strollo. Bernard Cosell and Eric Roberts helped design the front-end protocol, and Martin Thrope built the Pluribus-MLC adaptor. Many other members of the Pluribus group also made significant contributions. We would like to thank Robert Brooks and Sally Schindler for their help with the manuscript.

REFERENCES

1. Roberts, L. G. and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," *AFIPS Conference Proceedings 36*, June 1970, pp. 543-549.
2. Heart, F. E., R. E. Kahn, S. M. Ornstein, W. R. Crowther and D. C. Walden, "The Interface Message Processor for the ARPA Computer Network," *AFIPS Conference Proceedings 36*, June 1970, pp. 551-567; also in *Advances in Computer Communications*, W. W. Chu (ed.), Artech House Inc., 1974, pp. 300-316; also in *Computer Communications*, P. E. Green and R. W. Lucky (eds.), IEEE Press, 1975, pp. 375-391.
3. Ornstein, S. M., F. E. Heart, W. R. Crowther, S. B. Russell, H. K. Rising and A. Michel, "The Terminal IMP for the ARPA Computer Network," *AFIPS Conference Proceedings 40*, June 1972, pp 243-254; also in *Advances in Computer Communications*, W. W. Chu (ed.), Artech House Inc., 1974, pp. 317-328; also in *Computer Communications*, P. E. Green and R. W. Lucky (eds.), IEEE Press, 1975, pp. 354-365.
4. Heart, F. E., S. M. Ornstein, W. R. Crowther and W. B. Barker, "A New Minicomputer/Multiprocessor for the ARPA Network," *AFIPS Conference Proceedings 42*, June 1973, pp. 529-537; also in *Advances in Computer Communications*, W. W. Chu (ed.), Artech House Inc., 1974, pp. 329-337; also in *Computer Communication Networks*, R. L. Grimsdale and F. F. Kuo (eds.), *Proceedings of the NATO Advanced Study Institute* of September 1973, Sussex, England, published by Noordhoff International Publishing, Leyden, the Netherlands, 1975, pp. 159-180; also in *Computer Communications*, P. E. Green and R. W. Lucky (eds.), IEEE Press, 1975, pp. 366-374.
5. Retz, D., J. Miller, J. McClurg and B. Schafer, *ELF System Programmers Guide*, September 1975, Speech Communication Research Lab Inc., Santa Barbara, CA.
6. Bouknight, W. J., G. R. Grossman and D. M. Grothe, "The ARPA Network Terminal System—A New Approach to Network Access," *Proceedings of the Third ACM/IEEE Data Communications Symposium*, St. Petersburg, Florida, November 1973, pp. 73-79.
7. Ornstein, S. M., W. R. Crowther, M. F. Kraley, R. D. Bressler, A. Michel and F. E. Heart, "Pluribus—A Reliable Multiprocessor," *AFIPS Conference Proceedings 44*, May 1975, pp. 551-559.
8. Butterfield, S. C., R. D. Rettberg and D. C. Walden, "The Satellite IMP for the ARPA Network," *Proceedings of the Seventh Annual Hawaii International Conference on System Sciences*, Honolulu, Hawaii, January 1974, Computer Nets Supplement, pp. 70-73.
9. *Interface Message Processors for the ARPA Computer Network*, QTR No. 1, BBN Report No. 3063, April 1975.
10. *Specifications for the Interconnection of a Host and an IMP*, BBN Report No. 1822, January 1976, Appendix H.
11. Private Communication, D. C. Walden, January 1976.
12. Bobrow, D. G., J. D. Burchfiel, D. L. Murphy and R. S. Tomlinson, "TENEX, a Paged Time Sharing System for the PDP-10," *Communications of the ACM*, Volume 15, Number 3, March 1972, pp. 135-143.
13. Mimno, N. W., B. P. Cosell, D. C. Walden, S. C. Butterfield and J. B. Levin, "Terminal Access to the ARPA Network—Experience and Improvements," *Proceedings of the Seventh Annual IEEE Computer Society International Conference*, San Francisco, California, February 1973, pp. 39-43.
14. Walden, D. C., "Host-to-Host Protocols," *International Computer State of the Art Report No. 24:* Network Systems and Software, Infotech, Maidenhead, England, pp. 287-316.
15. *TIP Hardware Manual*, BBN Report No. 2184, November 1974.
16. *ARPA Network Current Network Protocols*, Stanford Research Institute, Menlo Park, California, December 1974.
17. *Interface Message Processors for the ARPA Computer Network*, QTR No. 2, BBN Report No. 3106, July 1975.
18. Cosell, B. P., P. R. Johnson, J. H. Malman, R. E. Schantz, J. Sussman, R. H. Thomas and D. C. Walden, "An Operational System for Computer Resource Sharing," *Proceedings of the Fifth Symposium on Operating Systems Principles*, November 1975, pp. 25-80.
19. *Specifications for the Interconnection of a Host and an IMP*, BBN Report No. 1822, January 1976, Section 3.8.
20. *The Remote Job Entry Mini-Host*, BBN Technical Information Report No. 93, August 1974.

Design issues for mixed media packet switching networks*

by D. HUYNH, H. KOBAYASHI** and F. F. KUO†
University of Hawaii
Honolulu, Hawaii

ABSTRACT

In this paper we present some of the important design issues for packet switching networks with both satellite and terrestrial components—which we call *mixed media* packet switching networks. Satellite packet switching has considerable promise for low cost, high bandwidth data communications. However there is inherent high delay in satellite links which does not appear in ground links. Therefore a mix of the two communications media offers the advantages of low-cost/high bandwidth together with low-delay communications where required. In this paper we examine a number of tradeoffs which offer guidelines for the design and optimum utilization of mixed media networks.

INTRODUCTION

In 1968 the Advanced Research Projects Agency (*ARPA*) of the U. S. Department of Defense began implementation of a computer-communication network which permits the interconnection of heterogeneous computers at geographically distributed centers throughout the United States. This network has come to be known as the *ARPANET*,[1,2] and has grown from the initial four node configuration in 1969 to almost forty nodes (including satellite nodes in Hawaii, Norway, and London) in late 1974. The major goal of ARPANET is to achieve resource sharing among the network users. The resources to be shared include not only programs, but also unique facilities such as the powerful ILLIAC IV computer and large global weather data bases that are economically feasible when widely shared.

The ARPANET employs a distributed store-and-forward packet-switching approach that is much better suited for computer-communication networks than the more conventional circuit-switching approach. Reasons favoring packet switching include lower cost, higher capacity, greater reliability and minimal delay. The CCITT (Comité Consultatif International Télégraphique et Téléphonique), an international standards organization, defines a packet as "a group of binary digits including data and call control signals which is switched as a composite whole. The data, call control signals and possibly error control information are arranged in a specific format."

Multi-access satellite channels

Communication satellites operating in packet mode are becoming increasingly important for consideration in the design of large computer communication networks. In particular, most possess the following characteristics which are of special importance: the satellite's antenna coverage allows any of a large number of ground stations to access it at any time (multi-access), and its transmissions can be received by all of these stations at all times (broadcast).

Up to the present, packet switched computer networks have mainly utilized terrestrial communications links. Recently, Telenet Communications Corporation, one of the new value-added carriers[3] announced plans to offer public packet switched data service in which terrestrial and satellite links will be available. *ARPA* has also planned to augment its terrestrial links with satellite communications and has commissioned several satellite IMPs or SIMPs[4] built by Bolt Beranek and Newman Inc. (BBN). In mid 1975, BBN and the British Post Office jointly conducted a multi-access satellite experiment using the Atlantic INTELSAT IV satellite with SIMPs located at the ETAM ground station in the U.S. and the Goonhilly Downs ground station in England.

The basis of this experiment is the work on the

* This report was supported by THE ALOHA SYSTEM, a research project at the University of Hawaii, which is supported by the Advanced Research Projects Agency of the Department of Defense and monitored by NASA Ames Research Center under Contract No. NAS2-8590 and by ONR Contract No. N00014-76-C-0256.

** Dr. Kobayashi is on leave from IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.

† Dr. Kuo is now with the Department of Defense, Washington, D.C., 20301.

ALOHANET of THE ALOHA SYSTEM project at the University of Hawaii.[5,6,7] The multiplexing technique that is used by the ALOHANET is a multi-access packet switching method that has come to be known as the *ALOHA* technique.

Based upon the ALOHA multiplexing method or variations thereof, a number of techniques have been proposed for the utilization of a satellite channel in a packet-switched data network in a way which allows all stations to dynamically share the channel capacity.[8,9,10,11,12,13,14] All are based on the division of the channel into time slots approximately equal to a packet transmission time. The approach described by Roberts[8] and Kleinrock and Lam in References 12 and 13 makes use of a technique called "slotted ALOHA random access" or "slotted ALOHA." The channel capacity of a slotted ALOHA channel was estimated to be $1/e \simeq 36\%$.[8]

Goals of this paper

In this paper we will examine a number of key issues in the design of a packet-switched communication network composed of a terrestrial store-and-forward packet switching component combined with a multi-access/broadcast satellite as depicted in Figure 1.

The factors involved in the design optimization of a packet switched network are:[15]

- Node location and traffic matrix
- Topology of links
- Capacity of links
- Routing
- Flow control, other network design, etc.

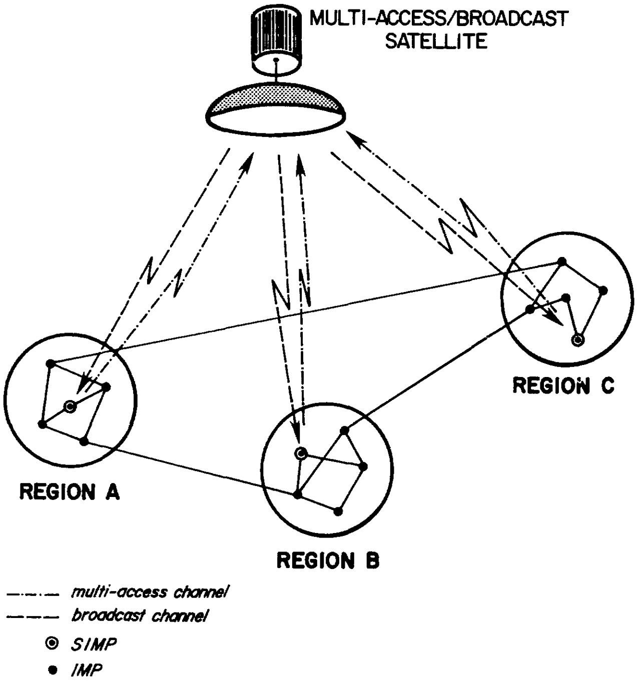


Figure 1—The proposed network model

To attempt to optimize any one factor, such as the capacity of links, it is necessary to assume that the factors higher in the list must be given. However, to decide optimum choices for the factors high on the list may require detailed calculations of all the lower ones. Fortunately, such factors as node location are usually not design variables, being fixed by practical considerations. Frank, Frisch and Chou[16] have developed sub-optimum procedures for the topological layout of a packet switched network. Thus we will concentrate on the following problems in our paper.

1. Routing of packets via ground or satellite
2. Capacity assignments for ground and satellite channels
3. Retransmission strategies

In this paper we will concentrate on the discussion of design issues such as throughput, delay, cost, etc., rather than dwell on the theoretical development of the design equations which is given in a companion paper to be published elsewhere.[17]

THE NETWORK MODEL

The network model under consideration consists of a terrestrial store-and-forward packet switching network referred to here as the *ground subnet*, and a multi-access/broadcast satellite which together with the SIMP ground stations is the *satellite subnet.*

For the ground subnet we will use the model given in the paper by Crowther, et al,[18] and will reproduce the succinct definitions of network terminology given in that paper:

Nodes—The nodes of the network are real-time computers, with limited storage and processing resources, which perform the basic packet-switching functions.

Hosts—The Hosts of the network are the computers, connected to nodes, which are the providers and users of the network services.

Lines—The lines of the network are some type of communications circuit of relatively high bandwidth and reasonably low error rate.

Connectivity—We assume a general, distributed topology in which each node can have multiple paths to other nodes, but not necessarily to all other nodes. Simple networks such as stars or rings are degenerate cases of the general topology we consider.

Message—The unit of data exchanged between source Host and destination Host.

Packet—The unit of data exchanged between adjacent nodes.

Acknowledgment—A piece of control information returned to a source to indicate successful receipt of a packet or message. A packet acknowledgment may be returned from an adjacent node to indicate successful receipt of a packet; a message acknowledgment may be returned from the destination to the source to indicate successful receipt of a message.

Store and Forward Subnetwork—The node stores a copy of a packet when it receives one, forwards it to an adjacent node, and discards its copy only on receipt of an acknowledgment from the adjacent node, a total storage interval of much less than a second.

Packet Switching—The nodes forward packets from many sources to many destinations along the same line, multiplexing the use of the line at a high rate.

Routing Algorithm—The procedure which the nodes use to determine which of the several possible paths through the network will be taken by a packet.

For the satellite subnet we use the finite population model of the slotted ALOHA channel developed by Kleinrock and Lam,[13] which we call *Scheme 1* and we also consider a multi-access channel model in which there is no retransmission via satellite. This model which we denote as *MASTER* or *Scheme 2* is described in a later section of this paper.

The combination of a system that operates in contention mode (the satellite) and one that operates in queueing mode (the terrestrial store-and-forward net) into an overall system model presents many interesting problems. These problems are mainly due to the fact that with contention systems the throughput increases to a maximum and then decreases as system load increases while for queueing systems the throughput increases to one as system load increases.[13] The model of a multi-access slotted-ALOHA channel is typical of a contention model, and to use the channel optimally the system load on the channel must be carefully controlled. We have tried to develop an analytical model of the overall system so that we may predict and optimize its performance. Our network model thus consists of:

A. A set of store-and-forward IMP-like devices interconnected by capacity limited ground channels (a distributed subnet). For the sake of reliability this subnet is at least 2-connected.

B. A set of SIMP-like devices directly connected to satellite ground stations. These SIMPs are usually geographically scattered and relatively far apart from each other.

C. A multi-access/broadcast satellite transponder linking all SIMPs in a star configuration.

A SIMP is usually co-located with an IMP at a node and both devices have buffering and scheduling capabilities. We assume that the ground net is regionalized. That is, the ground net is partitioned into regions. Every region has a SIMP and the IMPs in that particular region can only go through one and only one SIMP. The regionalization is determined by the closeness of an IMP to a SIMP in terms of the number of hops and the distance between them. Such a structure is shown in Figure 1.

ROUTING

With a mixed media network the issue of routing is a major concern. With two possible courses to choose from—one via satellite and one via ground, the issue is to choose the set of routes so as to minimize the overall average network delay. The tradeoffs to consider are these: the satellite channel has an inherent minimum delay of .26 seconds. However, satellite capacity is less costly than ground channel capacity for medium to long distances, and therefore more satellite capacity is available at less cost than comparable facilities on the ground. The ground channels are inherently faster than the satellite channel, but because of capacity limitations, are subject to queueing delays, which combined with the store-and-forward nodal processing delays may, in heavy traffic situations, result in larger overall delays than the satellite delays. To summarize, satellite channels have greater delays but also more cost effective channel bandwidth than ground channels.

The routing procedure used in the ARPANET is a distributed adaptive algorithm in which each node has a routing table which is periodically updated with minimum distance estimates from its immediate neighbors.[19] Unfortunately the distributed adaptive algorithm is extremely difficult to describe analytically and could not be applied to our analytical model. Instead we have chosen a deterministic split traffic (bifurcated) routing strategy [19] which because it allows traffic to flow on more than one path between a given source-destination node pair gives a better balance than a fixed routing procedure. It should be noted that our analytical model does not require any specific routing algorithm, but can accommodate any routing algorithm that can be modelled mathematically.

Optimal routing of packets

Using the network model of Figure 1 the main problem in packet routing is to route packets from one region to another via either ground or satellite links in such a way as to minimize the overall average delay of the network. Given the topology of the network and the capacities of the ground and satellite links, we assume a demand matrix $[\gamma_{ij}]$ where γ_{ij} is the average Poisson input in packets/sec from node i to node j. Let us define the *routing index* g_{ij} as the fraction of γ_{ij} sent through the ground net and let $\bar{g}_{ij}=1-g_{ij}$ be the fraction of γ_{ij} sent through the satellite net by first routing to the SIMP in the region where IMP i resides, then transmitting through the satellite channel to the SIMP in the region where IMP j resides. As a result, $g_{ij}=1$ if both IMPs i and j are in the same region.

Let us define T as being the overall average delay of the network, averaged over all source-destination node pairs and all routes dictated by the routing algorithm. The optimal routing strategy for mixed media networks can be stated as:

$$\min_{g_{ij}} T; \text{ subject to } \{0 \leq g_{ij} \leq 1 \text{ for all } i,j\}$$

For our analysis, we define the following notation:

$\gamma=\sum_{i,j}\gamma_{ij}$ total traffic in the net

τ_{ij}= average delay for a packet travelling from IMP i to IMP j

$\sigma(i)$ = index of the region (hence that of the corresponding SIMP) to which IMP i belongs

τ_s= average delay for a packet transmitted from one SIMP to another via the satellite channel

T_{ij}= overall average delay for a packet travelling from IMP i to IMP j (averaged over ground and satellite links)

We can readily derive the following relationships:

For the ALOHA scheme

$$T_{ij}=g_{ij}\tau_{ij}+\bar{g}_{ij}(\tau_{i\sigma(i)}+\tau_s+\tau_{\sigma(j)j}) \quad (1)$$

and

$$T=\frac{1}{\gamma}\sum_{i,j}\gamma_{ij}T_{ij}$$

$$=\frac{1}{\gamma}\sum_{i,j}\gamma_{ij}[g_{ij}\tau_{ij}+\bar{g}_{ij}(\tau_{i\sigma(i)}+\tau_{\sigma(j)j})]+\frac{1}{\gamma}\sum_{i,j}\bar{g}_{ij}\gamma_{ij}\tau_s \quad (2)$$

For the MASTER scheme, in order to obtain an expression analogous to Equations (1) and (2), we further introduce the notation:

P_σ= probability that a packet transmitted from SIMP σ into the satellite channel is successful

Then we obtain

$$T_{ij}=g_{ij}\tau_{ij}+\bar{g}_{ij}[\tau_{i\sigma(i)}+\tau_s+P_{\sigma(i)}\tau_{\sigma(j)j}+(1-P_{\sigma(i)})\tau_{\sigma(i)j}] \quad (3)$$

and

$$T=\frac{1}{\gamma}\sum_{i,j}\gamma_{ij}T_{ij}$$

$$=\frac{1}{\gamma}\sum_{i,j}\gamma_{ij}\{g_{ij}\tau_{ij}+\bar{g}_{ij}[\tau_{i\sigma(j)}+P_{\sigma(i)}\tau_{\sigma(j)j}$$

$$+(1-P_{\sigma(i)})\tau_{\sigma(i)j}]\}+\frac{1}{\gamma}\sum_{i,j}\bar{g}_{ij}\gamma_{ij}\tau_s$$

In Equations (2) and (4) the terms with g_{ij} represent the portion of traffic from IMP i to IMP j routed entirely through the ground net and those terms with $\bar{g}_{ij}$ represent the portion routed from node i to the regional SIMP $\sigma(i)$, from $\sigma(i)$ to $\sigma(j)$, the regional SIMP in which IMP j resides and then from $\sigma(j)$ to IMP j. Thus even in the satellite routing, there is some associated ground traffic to get to and from the regional SIMPs. Our mathematical model is flexible enough to permit SIMPs at every node, and if this were the case, obviously the $\tau_{i\sigma(i)}$ and $\tau_{\sigma(j)j}$ terms would be zero. The term τ_s is derived from the finite population model of a slotted ALOHA channel as originally given in Lam [13] and in a slightly different form by the authors.[17]

Let λ_l be the traffic on link l. We can derive λ_l as a function of g_{ij}, the traffic demand γ_{ij} and the routing algorithm. Since the equation is fairly complex we will not give it here but refer the reader to a companion paper for its derivation.[17] Given λ_l and neglecting the delay due to transmission errors and nodal processing and ground propagation time, the average delay along any ground link can be described by the M/M/1 queueing model by virtue of the well-known independence assumption and Jackson's decomposition theorem.[22]

$$T_l=\frac{1}{\mu_l C_l-\lambda_l} \quad (5)$$

where C_l is the capacity of channel l and $\frac{1}{\mu_l}$ the average packet length on that channel. The overall average delay can be expressed in terms of the link traffic λ_s, $\{\lambda_l\}$ and link delays τ_s, $\{T_l\}$ as

$$T=\frac{1}{\gamma}\left[\sum_{l=1}^{L}\lambda_l T_l+\lambda_s\tau_s\right] \quad (6)$$

where L = total number of links in the net, and λ_s the total traffic rate on the satellite channel, i.e.,

$$\lambda_s=\sum_i\sum_j\bar{g}_{ij}\gamma_{ij}$$

We can show that T in Equation (6) is convex with respect to g_{ij} so that we can obtain a minimum overall delay by varying g_{ij} according to a selected optimization algorithm. The optimum choice of g_{ij} depends also upon the topology of the network, the location of the SIMPs, the capacities of the ground and satellite links, the specific terrestrial routing algorithm, as well as the traffic demand matrix $[\gamma_{ij}]$, all of which must be specified in advance of the optimization computation.

Since the optimum selection of g_{ij} is not critically dependent upon a particular optimization procedure, we used the algorithm known as the BOX COMPLEX procedure [20] since this is one of the optimization algorithms which does not require derivatives of the objective function.

As an example, consider the network model in Figure 2 in which the two regions consist of nodes {1,2,3,4} and nodes {5,6,7,8}, and the regional SIMPs are located at nodes 1 and 7. The traffic demand matrix assumed to be uniform with $\gamma_{ij}=20$ packets/sec. $i \neq j$ and $\gamma_{ij}=0$

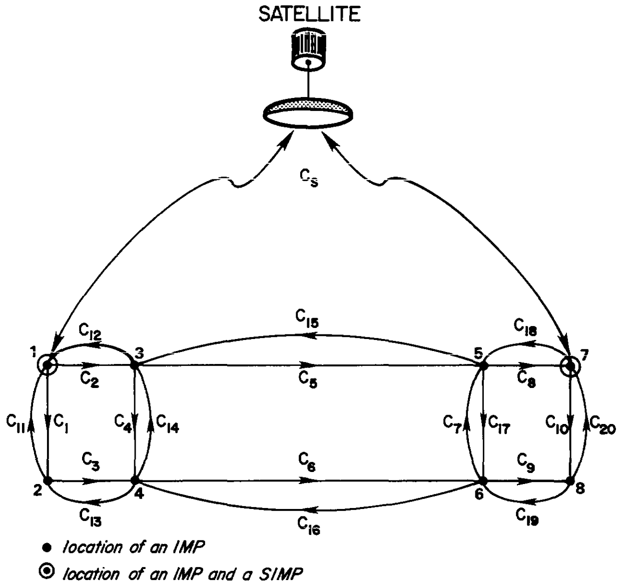


Figure 2—The network model considered in numerical examples

for $i=j$. The average packet lengths is assumed to be 512 bits on all ground channels. The packet length on the satellite channel is fixed and equals 1 K bit. The ground link capacities are all assumed to be $C_g=50$ K bits/sec and the satellite capacity to be $C_s=1500$ K bits/sec. With this information as input to the routing optimization program, the inter-regional g_{ij} are computed and are given in Table I. Note that we assume $g_{ij}=1$, for i,j in the same region.

LINK CAPACITY ASSIGNMENTS

In the packet routing studies above we assume that the ground link capacities are given. We also assume as given the capacity of the satellite channel. The capacity assignment problem is: for each channel l the capacity C_l must be found which minimizes the total average message delay T. The problem is most difficult if the capacities must be chosen from a discrete set of options. Following Kleinrock[21] we assume that the capacities of ground links and satellite channel are continuous variables and use analytic procedures involving Lagrange multipliers to obtain C_l. A general discussion of our approach follows.

TABLE I—The Routing Indices for Traffic from Region 1 to Region 2

		destination			
		5	6	7	8
origin	1	1	0	0	0
	2	.855	.681	0	.706
	3	1	1	0	.315
	4	1	1	0	1

Note: The Routing Indices for traffic from Region 2 to Region 1 are symmetrical to those given in Table 1. For instance, $g_{82}=g_{28}=.706$; $g_{72}=g_{18}=0$; etc.

We assume as given the topology of the network including SIMP locations. We also assume a given demand matrix $[\gamma_{ij}]$. Let λ_l be the traffic on link l. The (continuous) capacity assignment problem can be formulated as

$$\min_{C_l, C_s} \quad T=\frac{1}{\gamma}\left[\sum_{\ell=1}^{L}\lambda_l T_l+\lambda_s T_s\right]$$

subject to the constraint that the overall budget B is specified where B is

$$B=\sum_{\ell=1}^{L} b_l C_l+b_s C_s$$

and b_l and b_s are cost functions of the capacities C_l and C_s respectively.

In solving the capacity assignment problem, we need to know the total allowable budget and unit cost of ground and satellite channel capacities. We assume the unit cost of ground channel capacities are the same, so we can normalize all costs by a unit cost of ground channel capacity. In obtaining the following results, we further assume a ratio of *1* to *10* for satellite and ground channel capacity costs, i.e.,

$$\frac{1 \text{ unit cost of}}{\text{ground channel capacity}} = \frac{10 \text{ unit cost of}}{\text{satellite channel capacity}}$$

Our program is sufficiently general so that we can assume any ratio between satellite capacity and ground capacity costs. Perhaps a more realistic ratio to use today is a 1 to 3 proportion.

Without going into the details of the capacity assignment optimization scheme, which is quite similar to the work of Kleinrock[21] and which is described elsewhere,[17] let us discuss a computer program we have written which combines the routing and capacity assignment algorithms in an overall procedure. The program first obtains a solution for the routing assignments including overall average delay T_R, the routing indices g_{ij} and the channel traffic rates λ_s and λ_l. Then with the λ_s and λ_l from the routing calculations and with a given total budget B and the assumed unit costs of satellite and ground channel capacities b_s and b_l as inputs to the capacity assignment routine, we obtain the capacity assignments C_s and $\{C_l\}$ and a new figure for overall average delay T_C. If $T_C \leq T_R$ then the entire procedure is repeated with the previously computed values of C_s, $\{C_l\}$ and g_{ij} used as input to the routing subroutine.

The process is iterated until $T_C = T_R$ to within a specified accuracy or until the maximum time limit allowed for the optimization procedure is reached.

Using the output data from the routing subroutine run given in the previous section, the iterative cycle is carried out for the network in Figure 2. The input data consists of the original demand matrix, initial values of $C_l = 50$ K bits/sec and $C_s = 1500$ K bits/sec and an assumed total budget B in cost units and $b_l = 1$ and $b_s = 0.1$. The final results for $\{C_l, C_s\}$ and $\{\lambda_l, \lambda_s\}$ after 8 iterations for B = 1,700,000 cost units are given in Table II, and those for B = 950,000 cost units after 4 iterations are given in Table III. For the total budget B = 1,500,000 the minimum overall average delay was T = .037 sec. whereas for B = 950,000, T = 1.435 secs. We have performed a number of runs with different values of budget B and the results show that as total allowable budget increases the overall average delay decreases as seen in Figure 3. Also the optimum satellite capacity decreases and the ground channel capacities increase as the total allowable budget increases. This is because we have assumed a 10 to 1 ratio between the cost of unit ground capacity to unit satellite capacity. As the total budget grows we have more money to spend on the relatively more expensive ground channel capacities. As ground channel capacities become large, packets tend to go via ground subnet since the minimum satellite propagation delay is high (0.26 secs). We can show in general that the T vs B curve is a hyperbola and have derived formulas for the asymptote S which are given in a companion paper.[17]

TABLE II—Optimum Traffic Rates and Channel Capacities for B=1,700,000 Cost Units

		Optimum Traffic Rates (packets/sec) λ_l	*Optimum Capacities (K bits/sec)* C_l
Satellite Channel		2.15	15.966
Ground Channel	1	67.95	55.196
	2	120.06	88.238
	3	119.80	88.082
	4	92.43	70.967
	5	159.54	112.292
	6	159.43	112.226
	7	92.46	70.985
	8	120.00	88.201
	9	119.81	88.088
	10	67.80	55.097
	11	67.81	55.109
	12	120.09	88.259
	13	119.67	88.001
	14	92.46	70.984
	15	159.55	112.295
	16	159.33	112.163
	17	92.46	70.987
	18	120.00	88.203
	19	119.71	88.025
	20	67.90	55.165

The minimum delay=.0369 seconds

TABLE III—Optimum Traffic Rates and Channel Capacities for B=950,000 Cost Units

		Optimum Traffic Rates (packets/sec) λ_l	*Optimum Capacities (K bits/sec)* C_l
Satellite Channel		270.13	511.770
Ground Channel	1	93.61	47.405
	2	79.67	40.391
	3	88.29	44.730
	4	77.50	39.298
	5	81.71	41.417
	6	102.99	52.125
	7	77.61	39.351
	8	82.43	41.780
	9	86.86	44.009
	10	93.62	47.413
	11	91.26	46.222
	12	82.50	41.815
	13	85.94	43.547
	14	79.97	40.541
	15	82.07	41.598
	16	103.11	52.185
	17	79.97	40.539
	18	80.43	40.773
	19	89.34	45.257
	20	91.14	46.166

The minimum delay=1.435 seconds

THE MASTER SCHEME

In this section we will describe a scheme in which retransmission traffic from a multi-access channel is sent via the ground net. This scheme will be denoted here as the MASTER (**M**ulti-**A**ccess **S**atellite With **TE**rrestrial **R**etransmission) plan. Under the MASTER plan, the up-link of the satellite channel is a slotted multi-access channel and the down-link is a broadcast channel. Unlike slotted ALOHA, whenever a packet collision occurs in the multi-access channel, the MASTER scheme does not use the satellite channel for retransmission of the rejected packet. Instead the rejected packet is sent via the ground net from the source SIMP directly to the destination IMP without having to first go to its regional SIMP. By sending retransmitted packets via the ground net, overall average delay can be reduced significantly especially if the traffic on the satellite channel is heavy. Moreover, the MASTER plan eliminates all possibilities of instabilities that might occur in a slotted ALOHA channel under very heavy traffic situations.[13] Thus we see that the MASTER channel has no retransmitted packets. Since on the down-link broadcast channel each SIMP can hear its own transmissions, if a packet collision occurs, the source SIMP could reroute the rejected packet through the ground net without waiting for an acknowledgment from the destination SIMP. Moreover, after sending a packet, a SIMP can immediately send again in the next slot without waiting for the results of previous transmissions.

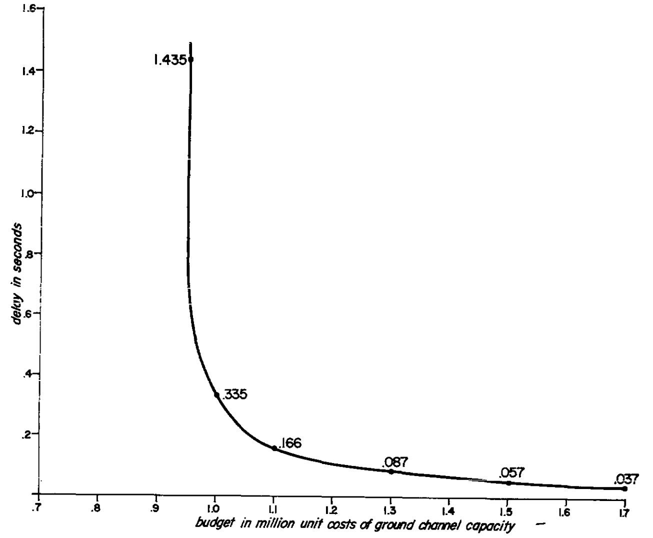


Figure 3—Delay-cost trade-offs

Now let us turn our attention to a more detailed discussion of the MASTER plan. Consider the satellite channel model depicted in Figure 4. Let λ_σ [pkts/sec] be the average input rate from SIMP σ to the satellite channel. The probability q_σ that a packet will be transmitted from SIMP σ in a given time slot, or equivalently the traffic rate in a given time slot is:

$$q_\sigma = \frac{\lambda_\sigma}{\mu_s C_s}$$

where C_s [bits/sec] is satellite channel capacity and $1/\mu_s$ [bits/sec] is the packet length.

We define random sequences $X_\sigma(k)$ and $Y_\sigma(k)$ as

$$X_\sigma(k) = \begin{cases} 1 \text{ if SIMP } \sigma \text{ transmits in time slot k} \\ 0 \text{ otherwise} \end{cases}$$

$$Y_\sigma(k) = \begin{cases} 1 \text{ if SIMP } \sigma \text{ successfully transmits in time slot k} \\ 0 \text{ if otherwise (i.e., no transmission or collision)} \end{cases}$$

Then it follows that

$$\Pr[X_\sigma = 1] = q_\sigma$$

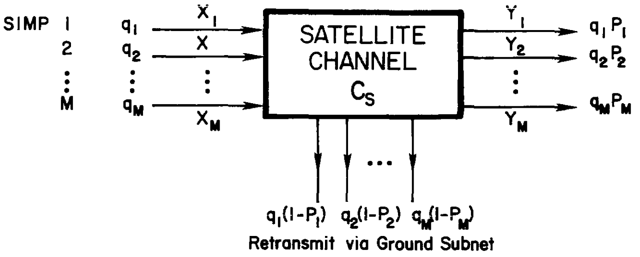


Figure 4—The satellite subnet model under Master

By assuming that the streams $X_\sigma(k)$ from different SIMPs are independent we readily obtain

$$\Pr[Y_\sigma = 1] = q_\sigma P_\sigma$$

where $P_\sigma = \prod_{\tau \neq \sigma} \bar{q}_\tau$, with $q_\tau = 1 - \bar{q}_\tau$, is the probability of success of a packet from SIMP σ. The normalized throughput S or successful transmission per slot time, of the satellite channel is thus

$$S = \sum_\sigma \Pr[Y_\sigma = 1]$$
$$= \sum_\sigma q_\sigma P_\sigma$$

Note that packets from SIMPs traverse the satellite channel once and only once under this operational scheme. If we assume the satellite channel transmission errors and nodal processing delay to be small, and buffer storage to be sufficiently large, then the average delay incurred by a packet when going through the channel can be found by using response time formulation of an M/D/1 queueing system plus the satellite propagation delay $\tau_{\min}$ of approximately .26 secs. The result of a deterministic service time system is used, since in a slotted channel all packets are of fixed length, any messages with length less than a specified full packet size are filled with blank characters to form a full size packet. Thus the average delay experienced by a packet travelling from SIMP σ through the satellite channel is

$$T_\sigma = \tau_{\min} + \frac{1}{\mu_s C_s}\left\{1 + \frac{\lambda_\sigma}{2(\mu_s C_s - \lambda_\sigma)}\right\}$$

The overall average packet delay of the satellite channel τ_s [secs/pkt] can be obtained by averaging over all T_σ's

$$\tau_s = \frac{1}{\lambda_s}\sum_\sigma \lambda_\sigma T_\sigma$$
$$= \tau_{\min} + \frac{1}{\mu_s C_s} + \frac{1}{2\lambda_s}\sum_\sigma \frac{q_\sigma^2}{1 - q_\sigma} \qquad (7)$$

where

$$\lambda_s = \sum_\sigma \lambda_\sigma$$

and

$$q_\sigma = \frac{\lambda_\sigma}{\mu_s C_s}$$

It can be shown that for a given total traffic rate $\lambda_s = \sum_\sigma \lambda_\sigma$ [pkts/sec] τ_s takes on its minimum when $\lambda_\sigma = \lambda_s/M$, for all $\sigma = 1,2,\ldots,M$. We can also show that the condition $\sum_\sigma q_\sigma = 1$ is necessary to achieve the maximum throughput S. From these conditions we immediately see that the best throughput delay trade-off is attained for $q_\sigma = 1/M$, $\sigma = 1,2,\ldots,M$.

This condition, although the optimum operating point, is difficult to attain, since the traffic rates from SIMPs are most likely not equal. It is difficult to study the trade-off for cases with a large number of heterogeneous SIMPs. We can obtain some understanding of how S varies by investigating the case of M=2. For this case, the throughput rate of the satellite channel is

$$S=q_1(1-q_2)+q_2(1-q_1)=q_1+q_2-2q_1q_2 \qquad (8)$$

A plot of the throughput S versus the traffic rates q_1 and q_2 is given in Figure 5. In the throughput as shown in Figure 5, the surface takes the form of a saddle with the valley located on the plane $q_1=q_2$ which includes the S-axis and is at a 45° angle from the planes $q_1=0$ and $q_2=0$ (the slashed, diagonal plane shown in Figure 5 and ridge located on the plane $q_1+q_2=1$ which is perpendicular to the plane $(q_1=q_2)$ and passes by upper-corner end points (1,0,1) and (0,1,1) of the throughput surface). Note that the condition

$$q_s=q_1+q_2 \qquad (9)$$

corresponds to a straight line of slope −1 on q_1-q_2 plane. As stated above if

$$q_1+q_2=q_s=1 \qquad (10)$$

we obtain the maximum boundary of the throughput S. The ridge of the saddle surface of the throughput S is thus the maximum boundary. For this simple case, we can further prove that, given the condition in Equation (9), S attains its minimum for

$$q_1=q_2 \qquad (11)$$

which is the line with slope 1 on q_1-q_2 plane (the line at 45° degree angle with q_1-axis and q_2-axis). The valley of the saddle surface of the throughput is thus the minimum boundary. Also, the overall average packet delay, τ_s, of the satellite channel for m=2 case is

$$\tau_s=\tau_{min}+\frac{1}{2\lambda_s}\left(\frac{q_1^2}{1-q_1}+\frac{q_2^2}{1-q_2}\right)+\frac{1}{\mu_s C_s} \qquad (12)$$

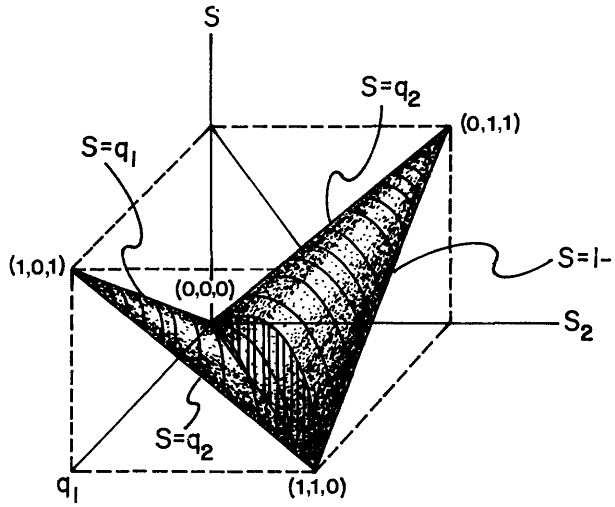


Figure 5—Throughput surface given by equation (8)

Again, the delay τ_s is symmetrical with respect to q_1 and q_2. In Figure 6 we show the level curves of the delay surface in the (q_1,q_2) plane. Given the condition in Equation (9) the delay τ_s reaches its minimum for $q_1=q_2$. The minima are shown by a dashed line intersecting the level curves.

The above results are preliminary. We have not computed numerical examples for the MASTER scheme. These will be given in a companion paper by the authors.[17]

CONCLUSIONS

In this paper we have presented some of the important design issues for mixed media packet switching networks. Satellite packet switching has considerable promise for low cost, high bandwidth data communications. However there is inherent high delay in satellite links which do not appear in ground links. Therefore a mix of the two communications media seems to offer the best of both worlds. In this paper we have examined a number of trade offs which offer guidelines for the design and optimum utilization of mixed media networks.

We have introduced a new communications scheme called MASTER (Multi-Access Satellite with TErrestrial Retransmission). We believe that the MASTER scheme offers significant advantages over slotted ALOHA especially when the multi-access satellite channel is heavily loaded.

We have not explored the possibility of sending network control information along the ground and using

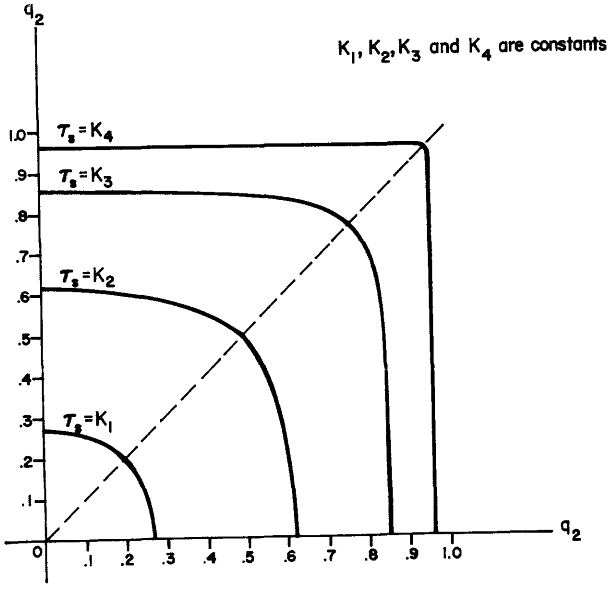


Figure 6—Level curves of delay surface given by equation (12)

the satellite for bulk data transmission. However the extension is logical and not difficult to analyze. Another logical extension of the MASTER scheme is to use the satellite channel on a reservation basis, such as suggested by Crowther, et al,[10] Roberts[11] and Binder,[14] but use the ground channel to set up reservations. We plan to explore this idea in a subsequent paper.

REFERENCES

1. Roberts, Lawrence G. and Barry D. Wessler, "The ARPA Network," in *Computer-Communication Networks*, edited by Norman Abramson and Franklin F. Kuo, Prentice-Hall, Inc., 1973, pp. 485-500.
2. Heart, F. E., "The ARPA Network," *Proceedings of the NATO Advanced Study Institute on Computer Communication Networks*, edited by R. L. Grimsdale and F. F. Kuo, Noordhoff International Publishers, 1975, pp. 19-34.
3. Wessler, Barry D. and Richard B. Hovey, "Public Packet-Switched Networks," *DATAMATION*, July 1974, pp. 85-87.
4. Butterfield, S., R. Rettberg, D. Walden, "The Satellite IMP for the ARPA Network," *Proceedings of the Seventh Hawaii International Conference on System Sciences—Subconference on Computer Nets*, Western Periodicals Co., January 1974, pp. 70-73.
5. Abramson, Norman, "The ALOHA System," in *Computer-Communication Networks*, edited by Norman Abramson and Franklin F. Kuo, Prentice-Hall, Inc., 1973, pp. 501-518.
6. Kuo, Franklin F. and Norman Abramson, "Some Advances in Radio Communications for Computers," *Digest of Papers —COMPCON '73*, San Francisco, February 1973, pp. 57-60.
7. Binder, Richard, et al, "ALOHA Packet Broadcasting—A Retrospect," *AFIPS National Computer Conference Proceedings*, Vol. 44, 1975, pp. 203-215.
8. Roberts, Lawrence G., "ALOHA Packet System With and Without Slots and Capture," *ARPANET Satellite System Note #8 (NIC 11290)*, Stanford Research Institute, June 1972.
9. Abramson, Norman, "Packet Switching with Satellites," *ALOHA SYSTEM Technical Report B73-2*, University of Hawaii, March 1973; also published in *AFIPS National Computer Conference Proceedings*, Vol. 42, New York, June 1973, pp. 695-702.
10. Crowther, W., et al, "A System for Broadcast Communications: Reservation ALOHA," *Proceedings of the Sixth Hawaii International Conference on System Sciences*, Western Periodicals Co., January 1973, pp. 371-374.
11. Roberts, Lawrence G., "Dynamic Allocation of Satellite Capacity Through Packet Reservation," *AFIPS National Computer Conference Proceedings*, Vol. 42, June 1973, p. 711.
12. Kleinrock, Leonard and Simon S. Lam, "Packet-Switching in a Slotted Satellite Channel," *AFIPS National Computer Conference Proceedings*, Vol. 42, June 1973, p. 703.
13. Lam, Simon S., "Packet-Switching in a Multi-Access Broadcast Channel with Application to Satellite Communication in a Computer Network," Ph.D. Thesis, UCLA; also available as *UCLA-ENG-7429*, UCLA Computer Science Department, April 1974.
14. Binder, Richard, "A Dynamic Packet Switching System for Satellite Broadcast Channels," *ALOHA System Technical Report B74-5*, University of Hawaii, August 1974.
15. Davies, D. W., "A Review of Computer Communications Technology," *Proceedings of the NATO Advanced Study Institute on Computer Communication Networks*, edited by R. L. Grimsdale and F. F. Kuo, Noordhoff International Publishers, 1975, pp. 1-17.
16. Frank, H., I. T. Frisch and W. Chou, "Topological Considerations in the Design of the ARPA Computer Network," *AFIPS Spring Joint Computer Conference Proceedings*, Vol. 36, 1970, pp. 543-549.
17. Huynh, D., H. Kobayashi and F. F. Kuo, "Optimal Design of Mixed Media Packet Switching Networks: Routing and Capacity Assignment" to be published in *IEEE Trans. on Communication*, Autumn 1976.
18. Crowther, W. R., F. E. Heart, A. A. McKenzie, J. M. McQuillan, and D. C. Walden, "Issues in Packet Switching Network Design," *AFIPS National Computer Conference Proceedings*, Vol. 44, 1975, pp. 161-175.
19. Fultz, G. L., "Adaptive Routing Techniques for Message Switching Computer-Communications Networks," Ph.D. Thesis, UCLA; also available as *UCLA-ENG-7252*, UCLA Computer Science Department, July 1972.
20. Box, M. J., "A New Method of Constrained Optimization and a Comparison with Other Methods," *The Computer Journal*, August 1965, p. 42.
21. Kleinrock, L., *Communication Nets: Stochastic Message Flow and Delay*, Dover Publications, New York, 1964.
22. Jackson, J. R., "Job Shop-like Queueing Systems," *Management Science*, Vol. 10, No. 1, October 1963, p. 131.

A perspective on network operating systems*

by STEPHEN R. KIMBLETON and RICHARD L. MANDELL
USC/Information Sciences Institute
Marina del Rey, California

ABSTRACT

The viability of packet switched computer communication has been demonstrated. The potential for more effective computing through resource sharing and load leveling is evident. Realization of this potential requires expanded user support to reduce or eliminate much of the need for users to learn the command languages of the hosts being accessed and of the communications subnetwork. Such a capability can be provided by a mediating agent providing ease of access to resources and control of resource access—a role traditionally ascribed to an operating system in the context of an individual computer system. This mediating agent, hereafter termed a Network Operating System (NOS), requires careful exploration to determine its appropriate interaction with the operating systems of the hosts within the network. This paper discusses the functions required of a Network Operating System and identifies major differences between the role of the Network Operating System and an individual host operating system. As such, it is intended to provide a basic perspective on the field of Network Operating Systems.

INTRODUCTION

Early computing experience quickly demonstrated the undesirability of requiring the user to directly interact with and control raw physical resources. As a result, operating systems were introduced which provided two primary functions: (i) provision of ease of access to computing resources, and (ii) control of the allocation of resources across multiple competing requests. Thus, the operating system may be regarded as an *agent* interposed between the user and the system resources.

Heterogeneous networks, such as ARPANET,[1-4] potentially permit both sharing of dissimilar resources and balancing of workload across a collection of similar resources such as the Operating System Class (OSC) consisting of the collection of TENEX operating systems within ARPANET. Presently, the viability of packet switched computer communications technology as used in ARPANET may be regarded as proved. However, the user wishing to make full use of the resource sharing potential of a heterogeneous network is currently faced with a requirement for learning the command languages of each separate system being accessed and, additionally, the command language required to support network communication. For the casual (non systems programmer) network user, this constitutes an immense startup cost which must be incurred prior to any reasonable use of network resources. The natural result is a minimization of the tendency to utilize potentially more effective remote resources.

Upon reflection, the obvious conclusion is that another entity, hereafter termed a mediator or *Network Operating System (NOS)*, is required to interface with the collection of host operating systems. In recognition of the urgent necessity for such a capability, an increasing number of papers are undergoing gestation and are scheduled to appear. The objective of this paper is to provide a global overview of the subject; to indicate some of the major components required for an NOS; to discuss the results of an NOS study[5] and to provide context for the remainder of the papers in this session.

It should be noted that the discussion of NOS in this paper is predicated upon an assumption that the existing host operating systems must remain essentially intact. In the alternative case in which one is free to design the operating systems for both the computer and the communications capabilities, substantial simplifications can result as the discussion in References 6 and 7 demonstrate.

To provide a perspective for reading subsequent sections, the remainder of this section identifies major differences between an NOS and an OS, establishes some of the major NOS objectives, and identifies requirements implicit in NOS design. The second section then discusses the major classes of primitives required for NOS implementation. Based upon this discussion, the two subsequent sections discuss issues and implica-

* Preparation of this paper was supported by the Rome Air Development Center, Distributed Computation Study, under Contract F30603-75-C-0222. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Rome Air Development Center, or the U.S. Government.

tions of the primitives required for data migration and network job execution. The fifth section examines the relationship of this work to existing capabilities and the sixth section closes with some summary remarks.

NOS-OS similarities and differences

From the viewpoint of an individual user, the basic atoms of both a network and an individual computer system are job steps and data. However, several major differences exist including: control of individual host resources; encapsulation; heterogeneity; geographical separation; and differing organizational constraints.

An individual operating system provides direct access to and control of individual host resources. In contrast, a Network Operating System provides a mediator which, to avoid substantial modifications to host operating systems, interfaces with the collection of operating systems to effect the requested functions. This has the advantage of eliminating the need for writing device drivers and many system utilities. The disadvantage is the need for explicit interfacing with a collection of distinct operating systems which may not provide comparable collections of capabilities.

A Network Operating System, in contrast with an individual operating system which has total control of system resources through effective encapsulation of the user, only mediates the interaction of the user with the operating system. Thus, the individual network user provided with access to a given system is assumed to have effectively the same potential spectrum of available capabilities as a local user. Moreover, since the user is not encapsulated, and the NOS is designed to support general purpose computing, it will usually be necessary to have some knowledge of local systems for interpretation of diagnostics and miscellaneous system responses created by program or programming errors. However, remote procedure calls, invocation of debugged programs, and many data operations could proceed without such knowledge.

For some applications, encapsulation may prove mandatory. Thus, the National Software Works provides such a capability in the context of providing a software production capability. This requires interrupt capturing, developing appropriate diagnostic translators and, for comparable items of software executing on distinct systems, development of standardized translators (grammars). This knowledge of the program to be executed is a prerequisite for its encapsulation. Encapsulation clearly provides significantly greater control of both user and host and, conceptually, is closer to heterogeneous multiprocessing. In view of the NSW demonstration of the feasibility of encapsulation, determination of its desirability is effectively a cost benefit exercise.

Host heterogeneity raises issues for both job processing and data movement. Job processing effectiveness is expedited through provision of a common command language.[8-10] However, as discussed above, determination of the proper amount of uniformity to provide in diagnostic and control messages may be regarded as an open issue. The data implication of host heterogeneity is reflected in the need for data selection, translation and transformation as discussed later.

The intent within a Network Operating System is to provide a collection of capabilities which enables an individual user to access remote resources (programs, data or systems) just as if the user were local to all these resources. Formally, this objective, although feasible, still requires caution on the part of the user in view of the time delays induced through accessing remote resources. This issue has been discussed in the context of providing a network based programming environment[11] and the care required seems similar to that required by the advent of virtual systems. That is, assuming that virtual memory was identical to real memory it could, for programs with poor locality, result in both poor utilization of system resources and high delay in program execution. It seems likely that improper utilization of network resources via an NOS will be reflected in the same manner.

Differing organizations have different viewpoints regarding sharing, accounting, and control of resource utilization. In a local environment these differences quickly become known and, through negotiation, become acceptable. In contrast, in a networking environment, the identity of the real user would usually be unknown and more formalized and rationalized procedures must be used. The precise implications of these organizational differences in style seem to be an open issue whose resolution is likely to be slow. However, the implications of poor accounting algorithms are already becoming manifest.[12-14]

NOS objectives and requirements

Five primary issues must be considered in developing a network operating system:

- provision of a uniform user viewpoint of resources
- modular expansibility
- control of host network interaction
- allocation of global network resources to network computing requirements, and
- implementation mode

The first objective was discussed above. The second is required to permit hosts to provide varying levels of NOS support as dictated by need. The third is required to assure adequate protection of host interests in a networking environment. The fourth is required to support efficient network based computing in general, and is mandatory for Mission Oriented Networks[5] in which a "collection" of computers must work cooperatively to achieve the organizational information processing function. The fifth consideration is required to ensure that the stated objectives do, indeed, prove economically feasible.

The developer of an operating system is effectively free to start from scratch. In contrast, the developer of a Network Operating System must interface with the existing software technology. Moreover, to be truly useful, the developer must also accept the fact that different sites are likely to have differing aspirations regarding network utilization and, as a result, will be likely to support different levels of sophistication in their NOS. It follows that a realistic NOS must be modularly expansible; the evident corollary is that it must support interaction with users not wishing to participate in the NOS as would be the case in ARPANET for users wishing to rely upon the existing protocols.

In almost any scientific or engineering endeavor, the first objective is to show that something can be done. The second is to show that it can be done well and the third is control. Control, in an NOS context, requires control of the host network interaction to prevent network demands from unfairly impacting local users. In addition, control in the context of a Mission Oriented Network[4] is also required to balance resource requirements against resource requests.

The mediator viewpoint described supports basic implementation and refinement of an NOS. The required element of control, however, imposes a host requirement significantly different from that currently existing since the network user is a relatively more unknown issue. As a result, the resource requirements of such a user are less predictable and the need for online, near real time control is increased. This, in turn, requires automated data gathering, archiving, and analysis. In addition, it requires the existence of a centralized network management capability to provide a common access point for users, individual installations and the network to obtain status and availability information. Such a capability would also permit balancing resource requirements against resources.

Host operating systems are complex, packet switched subnetworks are complex, and it is reasonable to assume that a completed network operating system will be complex. This necessitates careful consideration of its implementation mode. Two major alternatives exist: implementation within the host or implementation via augmentation of the host with a separate computer interposed between the host and the packet switch (IMP). In our opinion, the dramatic decline in hardware costs coupled with the increasing expense of sophisticated systems programmers argues strongly for the augmentation approach. Utilization of such an approach permits centralized design, implementation and support of the Network Operating Systems. Thus, only the host interfaces need be separately tailored. It should be noted that this support processor differs in scope from that usually subsumed under the title Front End Processor,[8] which is primarily concerned with offloading common portions of the software required to support ARPANET protocols. However, both have in common the need for case by case implementation of host interfaces.

NOS PRIMITIVE CATEGORIES

The global objectives of a Network Operating System require development of primitives in four categories:

- user communication,
- data migration,
- network job execution, and
- control.

User communication

User communication primitives are required since, in a geographically dispersed environment, the communication alternatives would require utilization of telephone, telegraph, etc. However, in view of the need to transmit programs, data, documentation, and user guidance (counseling), provision of a suitable mechanism within the computer communications network is clearly required. It should be noted that some of the more sophisticated individual host operating systems also provide such a capability.[15] Currently, rather general capabilities for user communication are available within the ARPANET for the TENEX subcollection of hosts.

In passing, it is worth noting that although rather general capabilities already exist, substantially more sophisticated capabilities can be considered. Two major branches can be distinguished: message processing and teleconferencing.

Message processing provides five basic categories of services:[16-18] creation, coordination, forwarding, alerting, and event processing. Through their provision, the ability of a large organization to respond in a timely manner to the dynamics of the environment is substantially facilitated. Creation and coordination permit generation and refinement of a proposed message by the collection of relevant originators. Forwarding encompasses the transmission process. Alerting supports early notification of the appropriate spectrum of recipients as determined from the addressee list or via content analysis. Finally, event processing can permit automatic invocation of computer operations appropriate to the message content (e.g., inventory reorders upon notification of outages in intermediate warehouses).

As observed in Reference 4, "Teleconferencing encompasses both multiparty voice/visual communication and a more formalized collection of capabilities related to hardcopy message communication.[19-23] A particular form of teleconferencing, computer-based conferencing, provides a natural support basis for communication among geographically dispersed computer users. This latter form of communication is also advantageous when it is appropriate to maintain permanent conference records, e.g., command and control. The general capabilities provided include archiving, indexing, searching, and updating of conferences. Through their utilization, communication between users is substantially enhanced."

Data migration

Data migration is the basic capability provided within a network for access to remote data. Current ARPANET provided capabilities primarily support transmission of sequential text files or block transmission of binary files. However, as discussed in the following section, efficient network utilization requires substantially more sophisticated capabilities. The spectrum of capabilities required for data migration is discussed later.

Network job execution

Network job execution differs from job execution in an individual computer system in the possibilities for: concurrent execution of parallel job steps; migration of a given job step to alternative sites; synchronization of job step execution across hosts; and the need for control of these capabilities. Moreover, in a heterogeneous network, job execution will generally require access to remote data. These and other issues are discussed in the next section.

Control

Control in a computer communications network can be considered at three levels:

- subnetwork control,
- host control, and
- network (host plus subnetwork) control.

Current subnetwork control capabilities are primarily limited to topology and link capacity modification. As a result, the control actions which can be invoked are rather static and the time constant for their invocation is on the order of months.

In the future, such static approaches to control promise to be unacceptable due to the emergent need for handling multimodal traffic including:[3,27]

- interactive traffic
- high throughput traffic such as file transfer,
- real time traffic such as digital transmission of speech, and
- guaranteed bandwidth traffic.

Since the available capacity directly interconnecting any two packet switches is relatively static, assurance of an equitable allocation of capacity among these traffic types as well as provision of guarantees that limited amounts of traffic of each type will get through requires careful investigation. An immediate requirement for such an investigation is online, near real time monitoring of traffic conditions and a likely corollary is an ability to effect control actions in near real time. The importance of research in this area is only beginning to be perceived with the transition of packet switching communication capabilities from the status of a research instrument to the status of a utility.

In a Value Added Network, by definition,[4] individual hosts as well as the subnetwork are organizationally independent. As a result, there is little need to coordinate and control the effects of job and data assignments. In contrast, in a Mission Oriented Network, a collection of hosts as well as, perhaps, the subnetwork are expected to work together cooperatively to achieve the organizational information processing function. This, in turn, is likely to require significant host control capabilities in order to ensure effective workload processing in the face of dynamically varying job and data assignments.

At the present time, work on centralized host network control capabilities is only beginning.[24] Their absence reflects the need for control strategies concerned with basic issues of: scheduling, major software capabilities and the basic hardware architecture of a system. In a single site computing environment, control strategies tended to be a byproduct of control tactics used for fine tuning the system.[25] Such tactics were primarily implemented by systems programmers having detailed knowledge of the system, its workload, anticipated needs, and the informal pecking order of customer priorities and were oriented toward a fine tuning of the system in a manner appropriate to such knowledge. It follows that tactics are relatively unsuited to centralized implementation while, as discussed in Reference 24, the opportunities for a centralized implementation of control strategies seem reasonable. A clear requirement for an effective control strategy is a careful, guaranteed delivery of system resources. An innovative approach to this subject is described in Reference 26.

Network (host plus subnetwork) control is clearly destined to be a subject of extreme importance in the context of Mission Oriented Networks. Existence of network control capabilities permits development and control of geographically dispersed data bases, tradeoffs between computing and communication, and assurance that remote users receive service comparable to that experienced by local users.

DATA MIGRATION

Data migration, as discussed in the Introduction, provides the basic capability required to permit a process executing in one computer to access data contained in another computer. Current data migration capabilities require that such access be accomplished via transmission of the file to the site at which the data is to be processed. Although this approach is, perhaps, reasonable in the context of an individual computer system, it seems unlikely to be acceptable in a data processing environment in which, typically, the owner of a file is unwilling to permit users to copy the file. In addition, for reasons of security and privacy, the owner may also be unwilling to let an accessing user access the entire file and may instead wish to restrict access to limited portions of the file.

Development of a general data migration capability requires three basic capabilities:

- data selection,
- data translation, and
- data transformation.

Data selection provides the basic capability (daemon) for remote accessing of data at the subfile level. As such, it should be envisioned as a transaction processor co-located with the data which, upon issuance of an appropriate request by a remote process, accesses a description of the file (specified in some appropriate data description language), and reads the requested record(s) in order to transmit them to the requesting process. At a conceptual level, it follows that the accessing process could therefore treat remote data just as it treats local data. At a practical level this observation needs to be tempered by the knowledge that network accesses will, in general, consume significantly more time than local data accesses and, as a result, access delays must be factored in to determine the relative desirability of utilizing a data selection capability versus transmission of the entire file. From an implementation viewpoint, it is of interest to observe that the basic data selection capabilities required relate closely to those required to support data translation.[28] Moreover, these capabilities also seem required to implement an effective locking capability in a networking environment.[29]

It is well recognized that hosts in a heterogeneous network use different bit patterns for encoding information. Data translation is the basic capability which permits hosts to communicate with each other in spite of these differences. It follows that a data translation capability is central to any effective capability to communicate among heterogeneous computers.

Data, particularly in a data processing environment, is usually stored as a collection of structured records. Effective transmission of data therefore requires augmentation of data translation to encompass preservation of record structure. The prerequisite capability for such preservation is an ability to describe the record structure and this, in turn, requires Data Description Languages.

Data description languages can be structured into Logical Data Description Languages (LDDLs) concerned with describing the logical structure of the data and with characterizing fields, and Physical Data Description Languages (PDDLs) which serve as the mechanism for describing how the data is physically laid out on the storage media.

LDDLs are relatively straightforward and can be pursued at various levels of abstraction as discussed in References 30, 31 and 32. PDDLs[33,34] are also logically straightforward but, from a user viewpoint, tedious. However, in a non-networking environment, PDDLs are clearly required to permit one system to read data written by another system. In contrast, in a networking environment, it seems feasible to eliminate the need for PDDLs and rely upon the system access routines to access the requested data and via a utility program, translate it into an appropriate "normal" form for transmission to the requesting site where it is again retranslated from the "normal" form into an appropriate local form. An exploratory discussion of this approach is contained in the paper: *An Approach To Data Migration in Computer Networks.*[28] It follows that data translation in a networking environment seems substantially simpler than translation in a non-networking environment.

Data transformation provides the basic capability for restructuring the logical form of data into a form more appropriate to the needs of the user or, alternatively, one which the owner of the data is willing to have transmitted. (For example, the owner may be unwilling to permit certain fields such as salary information to be transmitted.)

Data transformation requires, in general, three basic capabilities:

- arithmetic operations,
- logical operations,
- string operations.

Moreover, to be useful in a generalized data base environment, data transformation should permit operation on multiple input streams to produce multiple output streams.

Currently, some work is being done on the general topic of data transformation.[28,35,36] Moreover, a general single input stream single output stream data transformation capability was implemented within ARPANET in the context of the Data Reconfiguration Service. This system, in view of the primitive nature of the user interface and the highly procedural nature of the transformations could more properly be regarded as an experiment demonstrating the viability of the concept and, in this perspective should be viewed as a success. However, to be useful as a general purpose tool, substantially more sophisticated interface capabilities are required. Moreover, it would be generally desirable to permit both distributed and centralized implementations of this capability.

In summary, the basic needs required to support a data migration capability are reasonably clear and can be structured in a fairly coherent manner. Moreover, many of the required capabilities exist in various rudimentary forms at the present time.

NETWORK JOB EXECUTION

We define a network job to consist of a lattice (tied tree) of job steps which, as illustrated in Figure 1, may each be executed on a distinct and possibly dissimilar host. The characteristics of a network job are determined by those of the individual job steps.

In the simplest case, a job step may be a "batch" step which, once initiated, requires no further interaction with the NOS until termination. Support of the execution requirements of such a step is relatively straightforward.

In general, job steps executing in a networking environment may be anticipated to have more sophisticated support requirements including:

- user interaction,
- remote procedure calls, and
- synchronization with remote processes.

(It should be noted that execution of a network job is often viewed as an interprocess communication issue and thus a process rather than a job viewpoint is adopted. This viewpoint is correct for an implementor; however, from the viewpoint of the user the issue is really whether a remote job can be executed as is evident from the discussion of the Call-Return mechanism in Reference 11 as the basic primitive provided to a user.)

Support of user interaction with a network job step may prove relatively difficult in some operating systems since it tends to conflict with the desire for encapsulation of the job step which is useful in carefully controlling the interaction of the job step with the local host support capabilities such as the file system.

Provision of a capability to support remote procedure calls in a manner analogous to that in which local procedure calls are supported is clearly desirable. One approach to the development of such a capability is described in the paper *A High-Level Framework For Network-Based Resource Sharing.*[11] In this paper it is observed that this capability is particularly desirable to provide an alternative means to the more general issue of protocol implementation within a computer network.

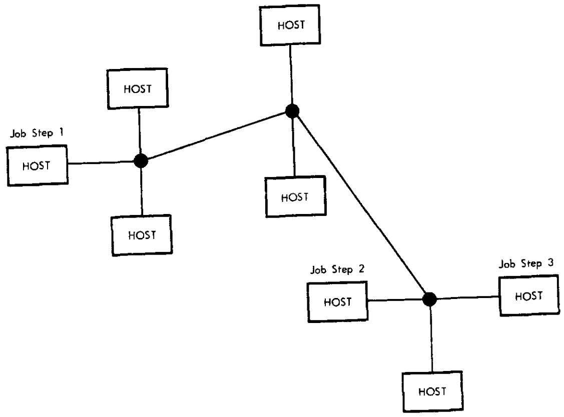


Figure 1—Job step assignment in a distributed network

Support of remote procedure calls carries, by analogy with the characteristics of local procedure calls, the implication that the calling process will terminate execution pending satisfaction of the call. In general, this need not be the case and, in view of the relatively large delays likely to occur in calls to remote systems (on the order of a few hundred milliseconds minimum due to subnetwork delays), more sophisticated capabilities are desirable to permit concurrent execution of local and remote processes. Thus, synchronization capabilities are required.

Network job execution requirements

Execution of a network job requires four major capabilities:

- job step assignment and control,
- step execution monitoring,
- JCL Generation, and
- Interprocess Communication Support.

Job step assignment and control is required for assigning job steps to processors: arranging for job step initiation upon satisfaction of all appropriate precedence and priority constraints; migration of files as required to permit job step initiation; and communication with the user. Assignment of job steps clearly requires a capability to detect any existing (step) parallelism. Moreover, such assignment may be made at the direction of the user, at the request of the host (to reduce peak workload), or at the suggestion of some centralized network management capability to balance global resource requests against resource availability.

Step execution monitoring is required to interface the scheduling of network originated jobs with local jobs. In particular, it provides step initiation; monitors step execution to provide restart and, potentially, recovery capabilities; and notifies the job step assignment routines of the termination of one step to permit initiation of successor steps. To support the needs of a job step to communicate with remote data, the appropriate data paths with data selectors stored in remote hosts must also be arranged.

Network JCL

A job control language provides four generic capabilities:

- identifying the precedence and priority conditions required for job step execution,
- making files within a user's directory known to

the step which usually has its own expectations regarding the naming of the files,

- inserting files generated by a program into the appropriate directory, and
- controlling the assignment of files to devices and, for a given file, controlling the layout of the file on the device.

Powerful job control languages, such as are commonly available with the larger data processing systems provide great power in effecting these functions and, as a result, are noted for the complexity of the statements required to achieve relatively simple (compile-load-go) objectives. In contrast, the JCL's provided with many of the scientifically oriented computer systems are quite simple and reflect a more limited spectrum of potential user needs. An intermediate point in this spectrum is provided by the capability which some of the more sophisticated JCL's provide of using "canned" procedures to achieve relatively straight-forward objectives. In this context, it seems likely that interactive approaches to JCL generation can be established which expedite generation of the appropriate JCL for the broad majority of user requirements. Users interested in maximum flexibility will probably still have to fend for themselves with the appropriate JCL. That is, user utilization of remote systems can be rendered no simpler than local usage if access to the full spectrum of system capabilities is to be provided.

Network IPC

Interprocess communication is required both as a technical prerequisite to building a network operating system and as a capability to be provided to users of the system to enable them to take full advantage of the opportunities (remote procedure calls, parallel execution, synchronized execution) afforded by the network. User IPC primitives are required for data transmission and process synchronization and control. Data primitives were discussed earlier. Synchronization and control primitives required to call procedures or processes, transfer data, and synchronize resource utilization can be divided into four major types:

- signal/wait-signal,
- wait/no-wait,
- transfer-control/retain-control, and
- preserve-state/reset-state.

The first primitive is required to permit a process to signal another process or procedure advertising its intentions; wait-signal is then required in order to permit a process to enter the wait state until receipt of an appropriate signal. The second primitive determines if parallel execution of both caller and called processes (entities) is to be supported The third primitive supports coroutines—in which both the calling and the called process have equal status—and the fourth capability is also required to support coroutine calls since the return location depends on the call location.

IPC capabilities can be provided in varying levels of sophistication reflecting: quality of the user interface, error recovery capabilities, and permissible complexity in the parameter string passed to the called procedure or process. Careful investigations of the total environment desirable to support effective and efficient IPC have been undertaken. Effectiveness issues are discussed in Reference 11, while issues of efficiency are discussed in Reference 37. Moreover, alternative message based implementation strategies to the connection oriented ARPANET approach to inter-process communication are described in Reference 38.

EXISTING RELATED CAPABILITIES

An overview of Network Operating Systems would not be complete without a discussion of currently existing capabilities. Currently, three such major capabilities can be distinguished:

- ARPANET Protocols,
- RSEXEC, and
- National Software Works.

ARPANET protocols

ARPANET protocols, as illustrated in Figure 2, can be structured in a hierarchical manner. The first level of user oriented protocols encompasses the three major function oriented protocols: TELNET, File Transfer Protocol (FTP), and Data Reconfiguration Service (DRS). The capabilities provided by these protocols have been discussed in the preceding sections and the salient factor from the viewpoint of an NOS is the observation that the protocol only standardizes the communication path between a user process (with which

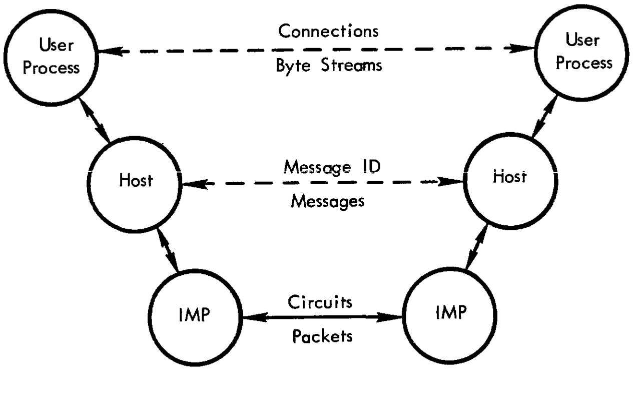


Figure 2—ARPANET protocol organization.[43,44] Solid lines denote real communication; dashed lines denote virtual communication

the user interacts) and a server process (which effects the necessary actions). Thus, these protocols at a minimum fail to provide program callability and require some degree of reprogramming for its achievement.

To provide more directly applicable user capabilities, ARPANET protocols also contain several applications oriented protocols such as Graphics[39] and the Network Voice Protocol[40] which are oriented toward provision of a specific collection of functional capabilities. As an alternative approach, one may wish to consider implementation of a more general user programming environment such as is proposed in the paper: *A High-Level Framework for Network-Based Resource Sharing.*[11]

RSEXEC

The Resource Sharing Executive (RSEXEC)[41] provides a network file system for the TENEX subcollection of hosts within the ARPANET. As such, it provides Network Wide Directories and, in addition, has supported system modifications which enable automatic file migration for dynamically generated file calls to non-local files. The standard set of inter-file manipulation commands are also provided. However, in view of the homogeneous nature of the hosts, issues of data translation did not require consideration and the issues of data selection, data transformation, and network job execution (excluding network IPC) were not specifically attacked.

National software works

The National Software Works (NSW)[42] is concerned with providing a program production capability distributed across a collection of hosts. It thereby potentially permits utilization of more sophisticated, relatively non-portable tools than would be possible if all tools were forced to execute at a single host.

Achievement of the NSW objectives requires careful consideration of many of the issues which are relevant in the design of a Network Operating System. However, in view of its more limited objective, substantially more sophisticated capabilities can be provided. In particular, in supporting tool access within the NSW framework, it proves feasible to render the host operating system apparently invisible to the user through effective encapsulation of the user and all interactions with the tool. Thus, the mediation role provided by an NOS is augmented to encompass encapsulation. In view of the magnitude of the NSW project we defer a more detailed discussion of NSW capabilities and refer the reader to Reference 5.

CONCLUDING REMARKS

In summary, we feel we have established the major objectives and goals underlying development of a general purpose Network Operating System. Although space limitations preclude a general discussion of their feasibility and manner of attainment, the interested reader may consult a study directed to this end.[5] As discussed therein, we feel it is reasonable to conclude that the objectives established are technically feasible; their utility is clearly manifest; and substantial portions of the required technological capabilities currently exist.

REFERENCES

1. Roberts, L. G. and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," in *Proceedings AFIPS 1970 Spring Joint Computer Conference,* Vol. 36, AFIPS Press, Montvale, New Jersey, 1970, pp. 543-549.
2. Roberts, L. G., "Data By The Packet," *IEEE Spectrum,* February 1974, pp. 46-51.
3. Walden, D. C., Personal Communication.
4. Kimbleton, S. R. and G. M. Schneider, "Computer Communication Networks: Approaches, Objectives, and Performance Considerations," in *Computing Surveys,* September 1975.
5. Kimbleton, S. R., Final Report, Contract F30603-75-C-0222, Rome Air Development Center, January 1976.
6. Hopwood, M. D., D. C. Loomis and L. A. Rowe, "The Design of a Distributed Computing System," Department of Information and Computer Science, University of California, Irvine, California, Technical Report 25, June 1973.
7. Mills, David L., *An Overview of the Distributed Computer Network,* University of Maryland, College Park, Maryland, 1976.
8. Padlipsky, Michael, "A Proposed Protocol for Connecting Host Computers to ARPA-like Network via Front-End Processors," MITRE Corporation, RFC 672, October 1974.
9. Rosenthal, Robert, "Accessing On-Line Network Resources with a Network Access Machine," *IEEE Intercon 1975,* IEEE, New York, 1975.
10. Uzgalis, R. C., "Four Languages Experiments in Computer Language Design," University of California, Los Angeles, California, July 1975.
11. White, James E., *A High-Level Framework for Network-Based Resource Sharing,* Stanford Research Institute, Menlo Park, California, 1976, (invited paper for session on Network Operating Systems at 1976 NCC).
12. Alsberg, P. A., "Distributed Processing on the ARPA Network—Measurements of the Cost and Performance Tradeoffs for Numerical Tasks," *Proceedings of the Eighth Hawaii International Conference on System Sciences,* Honolulu, Hawaii, January 1975, pp. 91-94.
13. Kimbleton, S. R., "Considerations in Pricing Distributed Computing," to appear in *Proceedings of the ACM Technical Meeting on Pricing Computing Services,* Palm Springs, California, November 1975.
14. *Proceedings of the ACM Technical Meeting on Pricing Computing Services,* Palm Springs, California, November, 1975.
15. Richie, D. M., D. Thompson, "The UNIX Time-Sharing System," *Communications of the ACM,* Vol. 17, No. 7, July, 1974, pp. 365-375.
16. Ellis, T. O., L. Gallenson, J. F. Heafner and J. T. Melvin, *A Plan for Consolidation and Automation of Military Telecommunications on Oahu,* ISI/RR-73-12, USC/Information Sciences Institute, Marina del Rey, California, May 1973.
17. Oestreicher, Donald, John Heafner, Jeffrey Rothenberg, "CONNECT, A User-Oriented Communications Service," in *Proceedings of the ACM, Annual Conference,* San Diego, California. November 1974, pp. 531-538.

18. Tugender, Ronald and Donald R. Oestreicher, *Basic Functional Capabilities for a Military Message Processing Service,* ISI/RR-75-23, USC/Information Sciences Institute, Marina del Rey, California, May 1975.
19. Carlisle, J. H., "A Selected Bibliography on Human Communication in Teleconferencing," Annotated, University of Southern California, Information Sciences Institute, August 1975.
20. ——— *A Tutorial for the Electronic Notebook Conference (TEN-C) System on ARPANET,* ISI/RR-75-38, USC/Information Sciences Institute, Marina del Rey, California, August 1975.
21. Turoff, M., "Delphi Conferencing: Computer-Based Conferencing with Anonymity," *Technological Forecasting and Social Change,* March 1972, pp. 159-204.
22. Vallee, J., H. M. Lipinski and R. H. Miller, *Group Communication Through Computers, Vol. 1: Design and Use of the FORUM System,* Report R-32, Institute for the Future, Menlo Park, California, July 1974.
23. Weston, J. R. and C. Kristen, "Teleconferencing: A Comparison of Attitudes Uncertainty and Interpersonal Atmospheres in Mediated and Face-To-Face Group Interaction," Report 1, The Social Policy and Programs Branch, The Department of Communications, Ottawa, Canada, December, 1973.
24. Kimbleton, Stephen R., "Computer System Design and Control: A Heuristically Based Approach," *Proceedings Second USA-Japan Computer Conference,* Tokyo, Japan, August 1975, pp. 264-270.
25. ———, "An Analytic Framework for Computer System Sizing and Tuning," *Proceedings of the NBS/ACM Workshop on Performance Analysis,* San Diego, March 1973, pp. 123-126.
26. Stefferud, E., D. L. Grobstein and R. Uhlig, "Wholesale/Retail Specialization in Resource Sharing Networks," *IEEE Computer,* Vol. 6, No. 8, August 1973, pp. 31-37.
27. Opderbeck, H. and L. Kleinrock, "The Influence of Control Procedures on the Performance of Packet-Switched Networks," *IEEE 1974 National Telecommunications Conference Record,* pp. 810-817.
28. Shu, Nan C., Vincent Y. Lum and Barron C. Housel, *An Approach to Data Migration in Computer Networks,* IBM Research Laboratory, 1976, IBM Research Report RJ1703.
29. Gray, J. N., R. A. Lorie and G. R. Putzolu, *Granularity of Locks in a Large Shared Data Base,* IBM Research Laboratory, June 30, 1975.
30. Codd, E. R., "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM,* Vol. 13, No. 6, ACM (Association for Computing Machinery), New York, London and Amsterdam, June 1970, pp. 377-387.
31. *Data Language/I—System/370 DOS/VS,* General Information Manual GH20-1246, IBM, White Plains, N.Y., 1974.
32. Martin, J., *Computer Data-Base Organizations,* Prentice-Hall, Englewood Cliffs, N.J., 1975.
33. Smith, D. P., "A Method for Data Translation Using the Stored Data Definition and Translation Task Group Languages," in *Proceedings of the 1972 ACM SIGFIDET Workshop on Data Description, Access and Control,* A. L. Dean, (Ed.), ACM, New York, 1972, pp. 107-124.
34. Merten, A. G. and J. P. Fry, "A Data Description Language Approach to File Translation," in *Proceedings of the 1974 ACM SIGMOD Workshop on Data Description, Access and Control,* Randall Rustin, (Ed.), ACM, New York, 1974, pp. 191-206.
35. Shoshani, Arie, "A Logical-Level Approach to Data Base Conversion," *ACM Sigmod,* International Conference on Management of Data, May 1975.
36. Schneider, G. Michael, "DSCL—A Data Specification and Conversion Language for Networks," *Proceedings of the ACM SIGMOD Conference,* San José, California, May 1975.
37. Sunshine, Carl, "Factors in Interprocess Communication Protocol Efficiency for Computer Networks," 1976 NCC Proceedings, Vol. 45.
38. Walden, D. C., "A System for Interprocess Communication in a Resource-Sharing Computer Network," *Communications of the ACM,* Vol. 15, No. 4, April 1972, pp. 221-230.
39. Michener, J., I. Cotton, K. Keiley, D. Liddle and E. Meyer, *Graphics Protocol,* Network Working Group, NIC No. 15358, April 1973.
40. Cohen, D., "Specifications for the Network Voice Protocol (NVP)," NCS Note 43 (Revision of NCS Notes 26 and 40), October 1974.
41. Thomas, Robert, "A Resource Sharing Executive for the ARPANET," *Proceedings of the AFIPS National Computer Conference,* Vol. 42, 1973, pp. 155-164.
42. Carlson, W. E. and S. D. Crocker, "The Impact of Networks on the Software Marketplace," in *Proc. Electronic and Aerospace Conference,* Washington, D.C., October 1974.
43. Metcalfe, R. M., *Packet Communication,* MAC TR-114, Massachusetts Institute of Technology, Project MAC, Cambridge, Massachusetts, December 1973.
44. Crocker, S. D., J. F. Heafner, R. M. Metcalfe and J. B. Postel, "Function-Oriented Protocols for the ARPA Computer Network," *AFIPS Conference Proceedings,* Spring Joint Computer Conference, 1972, pp. 271-280.
45. Cerf, Vinton G., Eric F. Harslem, John F. Heafner, Robert M. Metcalfe and James E. White, "An Experimental Service for Adaptable Data Reconfiguration," in *IEEE Transactions on Communications,* Vol. COM-20, No. 3, June 1972.
46. Retz, David L., Bruce W. Schafer, "Structure of the ELF Operating System," Stanford Research Institute, January 1976.

A high-level framework for network-based resource sharing*

by JAMES E. WHITE
Stanford Research Institute
Menlo Park, California

ABSTRACT

This paper proposes a high-level, application-independent framework for the construction of distributed systems within a resource sharing computer network. The framework generalizes design techniques in use within the ARPA Computer Network. It eliminates the need for application-specific communication protocols and support software, thus easing the task of the applications programmer and so encouraging the sharing of resources. The framework consists of a network-wide protocol for invoking arbitrary named functions in a remote process, and machine-dependent system software that interfaces one applications program to another via the protocol. The protocol provides mechanisms for supplying arguments to remote functions and for retrieving their results; it also defines a small number of standard data types from which all arguments and results must be modeled. The paper further proposes that remote functions be thought of as remotely callable subroutines or procedures. This model would enable the framework to more gracefully extend the local programming environment to embrace modules on other machines.

THE GOAL, RESOURCE SHARING

The principal goal of all resource-sharing computer networks, including the now international ARPA Network (the ARPANET), is to usefully interconnect geographically distributed hardware, software, and human resources.[1] Achieving this goal requires the design and implementation of various levels of support software within each constituent computer, and the specification of network-wide "protocols" (that is, conventions regarding the format and the relative timing of network messages) governing their interaction. This paper outlines an alternative to the approach that ARPANET system builders have been taking since work in this area began in 1970, and suggests a strategy for modeling distributed systems within any large computer network.

The first section of this paper describes the prevailing ARPANET protocol strategy, which involves specifying a family of application-dependent protocols with a network-wide inter-process communication facility as their common foundation. In the second section, the application-independent command/response discipline that characterizes this protocol family is identified and its isolation as a separate protocol proposed. Such isolation would reduce the work of the applications programmer by allowing the software that implements the protocol to be factored out of each applications program and supplied as a single, installation-maintained module. The final section of this paper proposes an extensible model for this class of network interaction that in itself would even further encourage the use of network resources.

THE CURRENT SOFTWARE APPROACH TO RESOURCE SHARING

Function-oriented protocols

The current ARPANET software approach to facilitating resource sharing has been detailed elsewhere in the literature.[2,3,4] Briefly, it involves defining a Host-Host Protocol by which the operating systems of the various "host" computers cooperate to support a network-wide inter-process communication (IPC) facility, and then various function-oriented protocols by which processes deliver and receive specific services via IPC. Each function-oriented protocol regulates the dialog between a resident "server process" providing the service, and a "user process" seeking the service on behalf of a user (the terms "user" and "user process" will be used consistently throughout this paper to distinguish the human user from the computer process acting on his behalf).

The current Host-Host Protocol has been in service since 1970. Since its initial design and implementation, a variety of deficiencies have been recognized and several alternative protocols suggested.[5,6] Although improvements at this level would surely have a positive effect upon Network resource sharing, the present paper simply assumes the existence of some form of

* The work reported here was supported by the Advanced Research Projects Agency of the Department of Defense, and by the Rome Air Development Center of the Air Force.

IPC and focuses attention upon higher level protocol design issues.

Each of the function-oriented protocols mentioned in this paper constitutes the official ARPANET protocol for its respective application domain and is therefore implemented at nearly all of the 75 host installations that now comprise the Network. It is primarily upon this widely implemented protocol family (and the philosophy it represents) that the present paper focuses. Needless to say, other important resource-sharing tools have also been constructed within the ARPANET. The Resource-Sharing Executive (RSEXEC), designed and implemented by Bolt, Beranek and Newman, Inc.,[7] provides an excellent example of such work.

Experience with and limitations of hands-on resource sharing

The oldest and still by far the most heavily used function-oriented protocol is the Telecommunications Network protocol (TELNET),[8] which effectively attaches a terminal on one computer to an interactive time-sharing system on another, and allows a user to interact with the remote system via the terminal as if he were one of its local users.

As depicted in Figure 1, TELNET specifies the means by which a user process monitoring the user's terminal is interconnected, via an IPC communication channel, with a server process with access to the target time-sharing system. TELNET also legislates a standard character set in which the user's commands and the system's responses are to be represented in transmission between machines. The syntax and semantics of these interchanges, however, vary from one system to another and are unregulated by the protocol; the user and server processes simply shuttle characters between the human user and the target system.

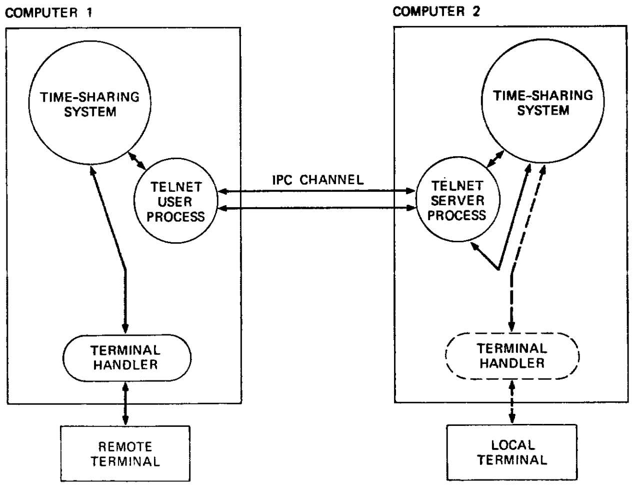


Figure 1—Interfacing a remote terminal to a local time-sharing system via the TELNET Protocol

Although the hands-on use of remote resources that TELNET makes possible is a natural and highly visible form of resource sharing, several limitations severely reduce its long-term utility:

(1) It forces upon the user all of the trappings of the resource's own system.

To exploit a remote resource, the user must leave the familiar working environment provided by his local system and enter an alien one with its own peculiar system structure (login, logout, and subsystem entry and exit procedures) and command language discipline (command recognition and completion conventions, editing characters, and so on). Hands-on resource sharing thus fails to provide the user with the kind of organized and consistent workshop he requires to work effectively.[9]

(2) It provides no basis for bootstrapping new composite resources from existing ones.

Because the network access discipline imposed by each resource is a human-engineered command language, rather than a machine-oriented communication protocol, it is virtually impossible for one resource to programatically draw upon the services of others. Doing so would require that the program deal successfully with complicated echoing and feedback characteristics; unstructured, even unsolicited system responses; and so forth. Hands-on resource sharing thus does nothing to provide an environment in which existing resources can be used as building blocks to construct new, more powerful ones.

These inherent limitations of hands-on resource sharing are removed by a protocol that simplifies and standardizes the dialog between user and server processes. Given such a protocol, the various remote resources upon which a user might wish to draw can indeed be made to appear as a single, coherent workshop by interposing between him and them a command language interpreter that transforms his commands into the appropriate protocol utterances.[10,11] The construction of composite resources also becomes feasible, since each resource is accessible by means of a machine-oriented protocol and can thus be readily employed by other processes within the network.

Standardizing the inter-machine dialog in specific application areas

After the TELNET protocol had been designed and widely implemented within the ARPANET, work began on a family of function-oriented protocols designed for use by programs, rather than human users. Each such protocol standardizes the inter-machine dialog in a particular application area. While TELNET dictates only the manner in which user and server processes are interconnected via the IPC facility, and the character set in which the two processes communicate once connected, each member of this family specifies in addition the syntax and semantics of the commands and responses that comprise their dialog.

Protocols within this family necessarily differ in substance, each specifying its own application-specific command set. The File Transfer Protocol (FTP),[12] for example, specifies commands for manipulating files, and the Remote Job Entry Protocol (RJE)[13] specifies commands for manipulating batch jobs. Protocols throughout the family are, however, similar in form, each successive family member having simply inherited the physical features of its predecessors. Thus FTP and RJE enforce the same conventions for formulating commands and responses.

This common command/response discipline requires that commands and responses have the following respective formats:

```
command-name    <SP> parameter <CRLF>
response-number <SP> text      <CRLF>
```

Each command invoked by the user process is identified by NAME and is allowed a single PARAMETER. Each response generated by the server process contains a three-digit decimal response NUMBER (to be interpreted by the user process) and explanatory TEXT (for presentation, if necessary, to the user). Response numbers are assigned in such a way that, for example, positive and negative acknowledgments can be easily distinguished by the user process.

FTP contains, among others, the following commands (each listed with one of its possible responses) for retrieving, appending to, replacing, and deleting files, respectively, within the server process file system:

Command	Response
RETR <SP> filename <CRLF>	250 <SP> Beginning transfer. <CRLF>
APPE <SP> filename <CRLF>	400 <SP> Not implemented. <CRLF>
STOR <SP> filename <CRLF>	453 <SP> Directory overflow. <CRLF>
DELE <SP> filename <CRLF>	450 <SP> File not found. <CRLF>

The first three commands serve only to initiate the transfer of a file from one machine to another. The transfer itself occurs on a separate IPC channel and is governed by what amounts to a separate protocol.

Since the general command format admits but a single parameter, multiparameter operations must be implemented as sequences of commands. Thus two commands are required to rename a file:

Command	Response
RNFR <SP> oldname <CRLF>	200 <SP> Next parameter. <CRLF>
RNTO <SP> newname <CRLF>	253 <SP> File renamed. <CRLF>

A COMMAND/RESPONSE PROTOCOL, THE BASIS FOR AN ALTERNATIVE APPROACH

The importance of factoring out the command/response discipline

That FTP, RJE, and the other protocols within this family share a common command/response discipline is a fact not formally recognized within the protocol literature, and each new protocol document describes it in detail, as if for the first time. Nowhere are these conventions codified in isolation from the various contexts in which they find use, being viewed as a necessary but relatively unimportant facet of each function-oriented protocol. "This common command/response discipline has thus gone unrecognized as the important, application-independent protocol that it is."

This oversight has had two important negative effects upon the growth of resource sharing within the ARPANET:

(1) It has allowed the command/response discipline to remain crude.

As already noted, operations that require more than a single parameter are consistently implemented as two or more separate commands, each of which requires a response and thus incurs the overhead of a full round-trip network delay. Furthermore, there are no standards for encoding parameter types other than character strings, nor is there provision for returning results in a command response.

(2) It has placed upon the applications programmer the burden of implementing the network "run-time environment (RTE)" that enables him to access remote processes at the desired, functional level.

Before he can address remote processes in terms like the following:

execute function DELE with argument TEXTFILE on machine X

the applications programmer must first construct (as he invariably does in every program he writes) a module that provides the desired program interface while implementing the agreed upon command/response discipline. This run-time environment contains the code required to properly format outgoing commands, to interface with the IPC facility, and to parse incoming responses. Because the system provides only the IPC facility as a foundation, the applications programmer is deterred from using remote resources by the amount of specialized knowledge and software that must first be acquired.

If, on the other hand, the command/response discipline were formalized as a separate protocol, its use in subsequent function-oriented protocols could rightly be anticipated by the systems programmer, and a single run-time environment constructed for use throughout an installation (in the worst case, one implementation per programming language per machine might be required). This module could then be placed in a library and, as depicted in Figure 2, link loaded with (or otherwise made available to) each new applications program, thereby greatly simplifying its use of remote resources.

Furthermore, since enhancements to it would pay dividends to every applications program employing its services, the run-time environment would gradually be augmented to provide additional new services to the programmer.

The thesis of the present paper is that one of the keys to facilitating network resource sharing lies in (1) isolating as a separate protocol the command/response discipline common to a large class of applications protocols; (2) making this new, application-independent protocol flexible and efficient; and (3) constructing at each installation a RTE that employs it to give the applications programmer easy and high-level access to remote resources.

Specifications for the command/response protocol

Having argued the value of a command/response protocol (hereafter termed the Protocol) as the foundation for a large class of applications protocols, there remains the task of suggesting the form that the Protocol might take. There are eight requirements. First, it must reproduce the capabilities of the discipline it replaces:

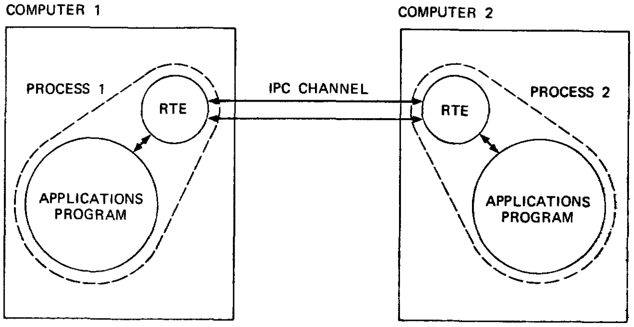


Figure 2—Interfacing distant applications programs via their run-time environments

(1) Permit invocation of arbitrary, named commands (or functions) implemented by the remote process.

(2) Permit command outcomes to be reported in a way that aids both the program invoking the command and the user under whose control it may be executing.

Second, the Protocol should remove the known deficiencies of its predecessor, that is:

(3) Allow an arbitrary number of parameters to be supplied as arguments to a single command.

(4) Provide representations for a variety of parameter types, including but not limited to character strings.

(5) Permit commands to return parameters as results as well as accept them as arguments.

And, finally, the Protocol should provide whatever additional capabilities are required by the more complex distributed systems whose creation the Protocol seeks to encourage. Although others may later be identified, the three capabilities below are recognized now to be important:

(6) Permit the server process to invoke commands in the user process, that is, eliminate entirely the often inappropriate user/server distinction, and allow each process to invoke commands in the other.

In the workshop environment alluded to earlier, for example, the user process is the command language interpreter and the server process is any of the software tools available to the user. While most commands are issued by the interpreter and addressed to the tool, occasionally the tool must invoke commands in the interpreter or in another tool. A graphical text editor, for example, must invoke commands within the interpreter to update the user's display screen after an editing operation.

(7) Permit a process to accept two or more commands for concurrent execution.

The text editor may wish to permit the user to initiate a long formatting operation with one command and yet continue to issue additional, shorter commands before there is a response to the first.

(8) Allow the process issuing a command to suppress the response the command would otherwise elicit.

This feature would permit network traffic to be reduced in those cases in which the process invoking the command deems a response unnecessary. Commands that always succeed but never return results are obvious candidates for this kind of treatment.

A formulation of the protocol that meets these specifications

The eight requirements listed above are met by a protocol in which the following two messages are defined:

message-type=COMMAND [tid] command-name arguments

message-type=RESPONSE tid outcome results

Here and in subsequent protocol descriptions, elements enclosed in square brackets are optional.

The first message invokes the command whose NAME is specified using the ARGUMENTS provided. The second is issued in eventual response to the first and returns the OUTCOME and RESULTS of the completed command. Whenever OUTCOME indicates that a command has failed, the command's RESULTS are required to be an error number and diagnostic message, the former to help the invoking program determine what to do next, the latter for possible presentation to the user. The protocol thus provides a framework for reporting errors, while leaving to the applications program the tasks of assigning error numbers and composing the text of error messages.

There are several elements of the Protocol that are absent from the existing command/response discipline:

(1) RESULTS, in fulfillment of Requirement 5.

(2) A MESSAGE TYPE that distinguishes commands from responses, arising from Requirement 6.

In the existing discipline, this distinction is implicit, since user and server processes receive only responses and commands, respectively.

(3) An optional transaction identifier TID by which a command and its response are associated, arising from Requirements 7 and 8.

The presence of a transaction identifier in a command implies the necessity of a response echoing the identifier; and no two concurrently outstanding commands may bear the same identifier.

Requirements 3 and 4—the ability to transmit an arbitrary number of parameters of various types with each command or response—are most economically and effectively met by defining a small set of primitive "data types" (for example, booleans, integers, character strings) from which concrete parameters can be modeled, and a "transmission format" in which such parameters can be encoded. Appendix A suggests a set of data types suitable for a large class of applications; Appendix B defines some possible transmission formats.

The protocol description given above is, of course, purely symbolic. Appendix C explores one possible encoding of the Protocol in detail.

Summarizing the arguments advanced so far

The author trusts that little of what has been presented thus far will be considered controversial by the reader. The following principal arguments have been made:

(1) The more effective forms of resource sharing depend upon remote resources being usefully accessible to other programs, not just to human users.

(2) Application-dependent protocols providing such access using the current approach leave to the applications programmer the task of constructing the additional layer of software (above the IPC facility provided by the system) required to make remote resources accessible at the functional level, thus discouraging their use.

(3) A single, resource-independent protocol providing flexible and efficient access at the functional level to arbitrary remote resources can be devised.

(4) This protocol would make possible the construction at each installation of an application-independent, network run-time environment making remote resources accessible at the functional level and thus encouraging their use by the applications programmer.

A protocol as simple as that suggested here has great potential for stimulating the sharing of resources within a computer network. First, it would reduce the cost of adapting existing resources for network use by eliminating the need for the design, documentation, and implementation of specialized delivery protocols. Second, it would encourage the use of remote resources by eliminating the need for application-specific interface software. And finally, it would encourage the construction of new resources built expressly for remote access, because of the ease with which they could be offered and used within the network software marketplace.

A HIGH-LEVEL MODEL OF THE NETWORK ENVIRONMENT

The importance of the model imposed by the protocol

The Protocol proposed above imposes upon the applications programmer a particular model of the network environment. In a heterogeneous computer network, nearly every protocol intended for general implementation has this effect, since it idealizes a class of operations that have concrete but slightly different equivalents in each system. Thus the ARPANET's TELNET Protocol alluded to earlier, for example, specifies a Network Virtual Terminal that attempts to provide a best fit to the many real terminals in use around the Network.

As now formulated, the Protocol models a remote resource as an interactive program with a simple, rigidly specified command language. This model follows naturally from the fact that the function-oriented protocols from which the Protocol was extracted were necessitated by the complexity and diversity of user-oriented command languages. The Protocol may thus legitimately be viewed as a vehicle for providing, as an adjunct to the sophisticated command languages already available to users, a family of simple command languages that can readily be employed by programs.

While the command/response model is a natural one, others are possible. A remote resource might also be modeled as a process that services and replies to requests it receives from other computer processes. This request/reply model would emphasize the fact that the Protocol is a vehicle for inter-process communication and that no human user is directly involved.

Substituting the request/reply model for the command/response model requires only cosmetic changes to the Protocol:

message-type=REQUEST [tid] op-code
arguments

message-type=REPLY tid outcome
results

In the formulation above, the terms "REQUEST", "REPLY", and "op-code" have simply been substituted for "COMMAND", "RESPONSE", and "command-name", respectively.

The choice of model need affect neither the content of the Protocol nor the behavior of the processes whose dialog it governs. Use of the word "command" in the command/response model, for example, is not meant to imply that the remote process can be coerced into action. Whatever model is adopted, a process has complete freedom to reject an incoming remote request that it is incapable of or unwilling to fulfill.

But even though it has no substantive effect upon the Protocol, the selection of a model—command/response, request/reply, and so on—is an important task because it determines the way in which both applications and systems programmers perceive the network environment. If the network environment is made to appear foreign to him, the applications programmer may be discouraged from using it. The choice of model also constrains the kind and range of protocol extensions that are likely to occur to the systems programmer; one model may suggest a rich set of useful extensions, another lead nowhere (or worse still, in the wrong direction).

In this final section of the paper, the author suggests a network model (hereafter termed the Model) that he believes will both encourage the use of remote resources by the applications programmer and suggest to the systems programmer a wide variety of useful Protocol extensions. Unlike the substance of the Protocol, however, the Model has already proven quite controversial within the ARPANET community.

Modeling resources as collections of procedures

Ideally, the goal of both the Protocol and its accompanying RTE is to make remote resources as easy to use as local ones. Since local resources usually take the form of resident and/or library subroutines, the possibility of modeling remote commands as "procedures" immediately suggests itself. The Model is further confirmed by the similarity that exists between local procedures and the remote commands to which the Protocol provides access. Both carry out arbitrarily complex, named operations on behalf of the requesting program (the caller); are governed by arguments supplied by the caller; and return to it results that reflect the outcome of the operation. The procedure call model thus acknowledges that, in a network environment, programs must sometimes call subroutines in machines other than their own.

Like the request/reply model already described, the procedure call model requires only cosmetic changes to the Protocol:

message-type=CALL [tid] procedure-name
arguments

message-type=RETURN tid outcome
results

In this third formulation, the terms "CALL", "RETURN", and "procedure-name" have been substituted for "COMMAND", "RESPONSE", and "command-name", respectively. And in this form, the Protocol might aptly be designated a "procedure call protocol (PCP)".

"The procedure call model would elevate the task of creating applications protocols to that of defining procedures and their calling sequences. It would also provide the foundation for a true distributed programming system (DPS) that encourages and facilitates the work of the applications programmer by gracefully extending the local programming environment, via the RTE, to embrace modules on other machines." This integration of local and network programming environments can even be carried as far as modifying compilers to provide minor variants of their normal procedure-calling constructs for addressing remote procedures (for which calls to the appropriate RTE primitives would be dropped out).

Finally, the Model is one that can be naturally extended in a variety of ways (for example, coroutine linkages and signals) to further enhance the distributed programming environment.

Clarifying the procedure call model

Although in many ways it accurately portrays the class of network interactions with which this paper deals, the Model suggested above may in other respects tend to mislead the applications programmer. The Model must therefore be clarified:

(1) Local procedure calls are cheap; remote procedure calls are not.

Local procedure calls are often effected by means of a single machine instruction and are therefore relatively inexpensive. Remote procedure calls, on the other hand, would be effected by means of a primitive provided by the local RTE and require an exchange of messages via IPC.

Because of this cost differential, the applications programmer must exercise discretion in his use of remote resources, even though the mechanics of their use will have been greatly simplified by the RTE. Like virtual memory, the procedure call model offers great convenience, and therefore power, in exchange for reasonable alertness to the possibilities of abuse.

(2) Conventional programs usually have a single locus of control; distributed programs need not.

Conventional programs are usually implemented as a single process with exactly one locus of control. A procedure call, therefore, traditionally implies a transfer of control from caller to callee. Distributed systems, on the other hand, are implemented as two or more processes, each of which is capable of independent execution. In this new environment, a remote procedure call need not suspend the caller, which is capable of continuing execution in parallel with the called procedure.

The RTE can therefore be expected to provide, for convenience, two modes of remote procedure invocation: a blocking mode that suspends the caller until the procedure returns; and a non-blocking mode that releases the caller as soon as the CALL message has been sent or queued. Most conventional operating systems already provide such a mode choice for I/O operations. For non-blocking calls, the RTE must also, of course, either arrange to asynchronously notify the program when the call is complete, or provide an additional primitive by which the applications program can periodically test for that condition.

Finally, the applications programmer must recognize that by no means all useful forms of network communication are effectively modeled as procedure calls. The lower level IPC facility that remains directly accessible to him must therefore be employed in those applications for which the procedure call model is inappropriate and RTE-provided primitives simply will not do.

SOME EXPECTATIONS

Both the Procedure Call Protocol and its associated Run-Time Environment have great potential for facilitating the work of the network programmer; only a small percentage of that potential has been discussed in the present paper. Upon the foundation provided by PCP can be erected higher level application-independent protocol layers that further enhance the distributed programming environment by providing even more powerful capabilities (see Appendix D).

As the importance of the RTE becomes fully evident, additional tasks will gradually be assigned to it, including perhaps those of:

(1) Converting parameters between the format employed internally by the applications program, and that imposed by the Protocol.
(2) Automatically selecting the most appropriate inter-process transmission format on the basis of the two machines' word sizes.
(3) Automatically substituting for network IPC a more efficient form of communication when both processes reside on the same machine.

The RTE will eventually offer the programmer a wide variety of application-independent, network-programming conveniences, and so, by means of the Protocol, become an increasingly powerful distributed-system-building tool.

ACKNOWLEDGMENTS

Many individuals within both SRI's Augmentation Research Center (ARC) and the larger ARPANET community have contributed their time and ideas to the development of the Protocol and Model described in this paper. The contributions of the following individuals are expressly acknowledged: Dick Watson, Jon Postel, Charles Irby, Ken Victor, Dave Maynard, and Larry Garlick of ARC; and Bob Thomas and Rick Schantz of Bolt, Beranek and Newman, Inc.

ARC has been working toward a high-level framework for network-based distributed systems for a number of years now.[14] The particular Protocol and Model described here result from research begun by ARC in July of 1974. This research included developing the Model; designing and documenting the Protocol required to support it;[15] and designing, documenting, and implementing a prototype run-time environment for a particular machine,[16,17] specifically a PDP-10 running the Tenex operating system developed by Bolt, Beranek and Newman, Inc.[18] Three design iterations were carried out during a 12-month period, and the resulting specification implemented for Tenex. The Tenex RTE provides a superset of the capabilities presented in the body of this paper and Appendices A through C as well as those alluded to in Appendix D.

REFERENCES

1. Kahn, R. E., "Resource-Sharing Computer Communications Networks," *Proceedings of the IEEE*, Vol. 60, No. 11, pp. 1397-1407, November 1972.
2. Crocker, S. D., J. F. Heafner, R. M. Metcalfe and J. B. Postel, "Function-oriented Protocols for the ARPA Computer Network," *AFIPS Proceedings*, Spring Joint Computer Conference, Vol. 40, pp. 271-279, 1972.
3. Carr, C. S., S. D. Crocker and V. G. Cerf, "Host-Host Communication Protocol in the ARPA Network," *AFIPS Proceedings*, Spring Joint Computer Conference, Vol. 36, pp. 589-597, 1970.
4. McKenzie, A. A., *Host/Host Protocol for the ARPA Network*, Bolt Beranek and Newman Inc., Cambridge, Massachusetts, January 1972, SRI-ARC Catalog Item 8246.
5. Walden, D. C., "A System for Interprocess Communication in a Resource Sharing Computer Network," *Communications of the ACM*, Vol. 15, No. 4, pp. 221-230, April 1972.
6. Cerf, V. G. and R. E. Kahn, "A Protocol for Packet Network Intercommunication," *IEEE Transactions on Communications*, Vol. Com-22, No. 5, pp. 637-648, May 1974.
7. Thomas, R. H., "A Resource-Sharing Executive for the ARPANET," *AFIPS Proceedings*, National Computer Conference, Vol. 42, pp. 155-163, 1973.
8. *TELNET Protocol Specification*, Stanford Research Institute, Menlo Park, California, August 1973, SRI-ARC Catalog Item 18639.
9. Engelbart, D. C., R. W. Watson and J. C. Norton, "The Augmented Knowledge Workshop," *AFIPS Proceedings*, National Computer Conference, Vol. 42, pp. 9-21, 1973.
10. Engelbart, D.C. and W. K. English, "A Research Center for Augmenting Human Intellect," *AFIPS Proceedings*, Fall Joint Computer Conference, Vol. 33, pp. 395-410, 1968.
11. Irby, C. H., C. F. Dornbush, K. E. Victor and D. C. Wallace, *A Command Meta Language for NLS*, Final Report, Contract RADC-TR-75-304, SRI Project 1868, Stanford Research Institute, Menlo Park, California, December, 1975.
12. Neigus, N. J., *File Transfer Protocol*, ARPA Network Working Group Request for Comments 542, Bolt Beranek and Newman Inc., Cambridge, Massachusetts, July 1973, SRI-ARC Catalog Item 17759.
13. Bressler, R. D., R. Guida and A. A. McKenzie, *Remote Job Entry Protocol*, ARPA Network Working Group Request for Comments 360, Dynamic Modeling Group, Massachusetts Institute of Technology, Cambridge, Massachusetts, undated, SRI-ARC Catalog Item 12112.
14. Watson, R. W., *Some Thoughts on System Design to Facilitate Resource Sharing*, ARPA Network Working Group Request for Comments 592, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, November 20, 1973, SRI-ARC Catalog Item 20391.
15. White, J. E., *DPS-10 Version 2.5 Implementer's Guide*, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, August 15, 1975, SRI-ARC Catalog Item 26282.
16. White, J. E., *DPS-10 Version 2.5 Programmer's Guide*, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, August 13, 1975, SRI-ARC Catalog Item 26271.
17. White, J. E., *DPS-10 Version 2.5 Source Code*, Augmentation Research Center, Stanford Research Institute, Menlo Park, California, August 13, 1975, SRI-ARC Catalog Item 26267.
18. Bobrow, D. G., J. D. Burchfiel, D. L. Murphy and R. S. Tomlinson, "TENEX, a Paged Time Sharing System for the PDP-10," *Communications of the ACM*, Vol. 15, No. 3, pp. 135-143, March 1972.
19. White, J. E., "Elements of a Distributed Programming System," Submitted for publication in the *Journal of Computer Languages*, 1976.

APPENDIX A—SUGGESTED DATA TYPES

The Protocol requires that every parameter or "data object" be represented by one of several primitive data types defined by the Model. The set of data types below is sufficient to conveniently model a large class of data objects, but since the need for additional data types (for example, floating-point numbers) will surely arise, the set must remain open-ended. Throughout the descriptions below, N is confined to the range [0, 2**15-1]:

LIST: A list is an ordered sequence of N data objects called "elements". A LIST may contain other LISTs as elements, and can therefore be employed to construct arbitrarily complex composite data objects.

CHARSTR: A character string is an ordered sequence of N ASCII characters, and conveniently models a variety of textual entities, from short user names to whole paragraphs of text.

BITSTR: A bit string is an ordered sequence of N bits and, therefore, provides a means for representing arbitrary binary data (for example, the contents of a word of memory).

INTEGER: An integer is a fixed-point number in the range [-2**31, 2**31-1], and conveniently models various kinds of numerical data, including time intervals, distances, and so on.

INDEX: An index is an integer in the range [1, 2**15-1]. As its name and value range suggest, an INDEX can be used to address a particular bit or character within a string, or element within a list. INDEXes have other uses as well, including the modeling of handles or identifiers for open files, created processes, and the like. Also, because of their restricted range, INDEXes are more compact in transmission than INTEGERs (see Appendix B).

BOOLEAN: A boolean represents a single bit of information, and has either the value true or false.

EMPTY: An empty is a valueless place holder within a LIST or parameter list.

APPENDIX B—SUGGESTED TRANSMISSION FORMATS

Parameters must be encoded in a standard transmission format before they can be sent from one process to another via the Protocol. An effective strategy is to define several formats and select the most appropriate one at run-time, adding to the Protocol a mechanism for format negotiation. Format negotiation would be another responsibility of the RTE and could thus be made completely invisible to the applications program.

Suggested below are two transmission formats. The first is a 36-bit binary format for use between 36-bit machines, the second an 8-bit binary, "universal" format for use between dissimilar machines. Data objects are fully typed in each format to enable the RTE to automatically decode and internalize incoming parameters should it be desired to provide this service to the applications program.

PCPB36, For Use Between 36-Bit Machines

Bits 0-13 Unused (zero)
Bits 14-17 Data type

EMPTY	=1	INTEGER	=4	LIST	=7
BOOLEAN	=2	BITSTR	=5		
INDEX	=3	CHARSTR	=6		

Bits 18-20 Unused (zero)
Bits 21-35 Value or length N

EMPTY	unused (zero)
BOOLEAN	14 zero-bits + 1-bit value (TRUE=1/FALSE=0)
INDEX	unsigned value
INTEGER	unused (zero)
BITSTR	unsigned bit count N
CHARSTR	unsigned character count N
LIST	unsigned element count N

Bits 36- Value

EMPTY	unused (nonexistent)
BOOLEAN	unused (nonexistent)
INDEX	unused (nonexistent)
INTEGER	two's complement full-word value
BITSTR	bit string + zero padding to word boundary
CHARSTR	ASCII string + zero padding to word boundary
LIST	element data objects

PCPB8, For Use Between Dissimilar Machines

Byte 0 Data type

EMPTY	=1	INTEGER	=4	LIST	=7
BOOLEAN	=2	BITSTR	=5		
INDEX	=3	CHARSTR	=6		

Bytes 1- Value

EMPTY	unused (nonexistent)
BOOLEAN	7 zero-bits + 1-bit value (TRUE=1/FALSE=0)
INDEX	2-byte unsigned value
INTEGER	4-byte two's complement value
BITSTR	2-byte unsigned bit count N + bit string + zero padding to byte boundary
CHARSTR	2-byte unsigned character count N + ASCII string
LIST	2-byte element count N + element data objects

APPENDIX C—A DETAILED ENCODING OF THE PROCEDURE CALL PROTOCOL

Although the data types and transmission formats detailed in the previous appendixes serve primarily as vehicles for representing the arguments and results of remote procedures, they can just as readily and effectively be employed to represent the commands and responses by which those parameters are transmitted.

Taking this approach, one might model each of the two Protocol messages as a PCP data object, specifically a LIST whose first element is an INDEX message type. The following concise statement of the Protocol then results:

```
LIST (CALL,       tid,      procedure,  arguments)
      INDEX=1   INDEX/
                EMPTY     CHARSTR     LIST
LIST (RETURN,     tid,      outcome,    results)
      INDEX=2   INDEX     BOOLEAN     LIST
```

The RESULTS of an unsuccessful procedure would be represented as follows:

```
LIST (error,    diagnostic)
      INDEX     CHARSTR
```

APPENDIX D—A LOOK AT SOME POSSIBLE EXTENSIONS TO THE MODEL

The result of the distributed-system-building strategy proposed in the body of this paper and the preceding appendices is depicted in Figure 3. At the core of each process is the inter-process communication facility provided by the operating system, which effects the transmission of arbitrary binary data between distant processes. Surrounding this core are conventions regarding first the format in which a few, primitive types of data objects are encoded in binary for IPC, and then the formats of several composite data objects (that is, messages) whose transmission either invokes or acknowledges the previous invocation of a remote procedure. Immediately above lies an open-ended protocol layer in which an arbitrary number of enhancements to the distributed programming environment can be implemented. Encapsulating these various protocol layers is the installation-provided run-time environment, which delivers DPS services to the applications program according to machine- and possibly programming-language-dependent conventions.

The Protocol proposed in the present paper recognizes only the most fundamental aspects of remote procedure calling. It permits the caller to identify the procedure to be called, supply the necessary arguments, determine the outcome of the procedure, and recover its results. In a second paper,[19] the author proposes some extensions to this simple procedure call model, and attempts to identify other common forms of inter-process interaction whose standardization would

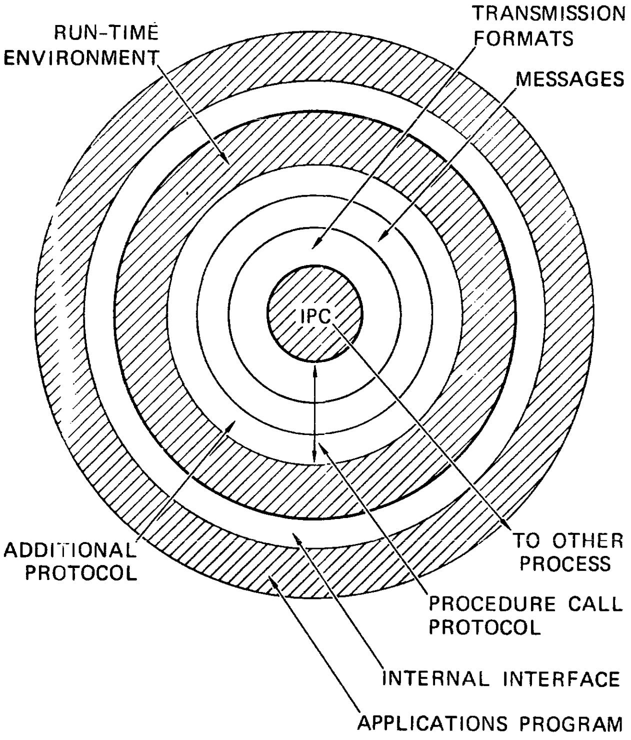


Figure 3—Software and protocol layers comprising a process within the distributed programming system

enhance the distributed programming environment. Included among the topics discussed are:

(1) Coroutine linkages and other forms of communication between the caller and callee.
(2) Propagation of notices and requests up the thread of control that results from nested procedure calls.
(3) Standard mechanisms for remotely reading or writing system-global data objects within another program.
(4) Access controls for collections of related procedures.
(5) A standard means for creating and initializing processes, that is, for establishing contact with and logging into a remote machine, identifying the program to be executed, and so forth. This facility would permit arbitrarily complex process hierarchies to be created.
(6) A mechanism for introducing processes to one another, that is, for superimposing more general communication paths upon the process hierarchy.

These and other extensions can all find a place in the open-ended protocol layer of Figure 3. The particular extensions explored in Reference 19 are offered not as dogma but rather as a means of suggesting the possibilities and stimulating further research.

Factors in interprocess communication protocol efficiency for computer networks*

by CARL A. SUNSHINE
The Rand Corporation
Santa Monica, California

ABSTRACT

This paper considers the efficiency of interprocess communication protocols for distributed processing environments such as computer networks. Previous research has emphasized system performance at lower levels, within the communication medium itself, while this work examines requirements and performance of protocols for communication between processes in the Host computers attached to the communication system. Efficiency primarily concerns throughput and delay achievable for communication between remote processes. Various aspects of protocol operation are analyzed, and protocol policies concerning retransmission, flow control, buffering, acknowledgment, and packet size emerge as the most important factors in determining efficiency. Several graphs showing quantitative performance results for representative situations are included.

INTRODUCTION

The tremendous growth of computer communications in recent years has provided new problems as well as new opportunities in distributed processing. In particular, computer networks such as the ARPANET[7,23] demand new techniques for providing reliable and efficient communication between processes running in different Host computers connected to the network. Typical network transmission characteristics include variable delay, limited bandwidth, and occasional loss, damage, duplication, or out-of-order delivery of messages.[1–3] These transmission characteristics demand specially robust *protocols*, or algorithms and message formats for data exchange between remote processes, compared to mechanisms appropriate in a centralized system with common memory.

Protocols for such environments are usually based on transmission of packets containing data, control information such as sequence numbers, and a checksum for error detection. Correctly received packets are positively acknowledged by the destination. If no acknowledgment is received at the source within a given time-out period, packets are retransmitted. The reliability of such Positive Acknowledgment, Retransmission (PAR) protocols in the face of network transmission characteristics mentioned above has received increasing attention.[3] This paper discusses major factors determining the *efficiency* of protocols for communication between remote processes over a packet switching network (PSN). Research on interprocess communication level protocols has just begun[4,5] while related work on performance analysis of protocols used within a PSN is more abundant.[6–11]

The two performance measures of primary interest are mean *delay* and mean *throughput* attainable. Throughput may be defined as the transmission rate of useful data between processes, excluding any control information or retransmissions required by the protocol. Delay is the time from starting to transmit a packet at the sender to successful arrival of the entire packet at the receiver in the case of one-way delay, or until arrival of an acknowledgment at the sender in the case of roundtrip delay. (Note that any waiting time between packet creation and start of transmission is not included in this delay definition.) Other efficiency performance measures of interest include *retransmission rate* (the proportion of packets which are retransmissions), *line efficiency* (the ratio of useful traffic to total traffic), and *buffer requirements.*

Numerous factors such as buffer allocation, receiver processing rate, flow control, error rates, error recovery techniques, header overhead, and network transmission delay and bandwidth help determine protocol performance. Some of these factors depend directly on communication network design and operation and represent essentially uncontrollable characteristics from the point of view of an interprocess communication protocol. Other factors depend on the behavior of processes communicating via the protocol.

* This work was supported by the Defense Advanced Research Projects Agency under ARPA Order No. 2494, Contract No. MDA903-76C-0093 with Stanford University. The views and conclusions contained in this document are those of the author and do not necessarily represent the policies of the Defense Advanced Research Projects Agency of the United States Government.

This paper focuses on a third class of factors which are subject to control by the protocol itself. To provide efficient communication within the constraints of given network characteristics and user process behavior, a protocol can attempt to optimize several internal parameters such as retransmission interval, flow control strategy, buffering, acknowledgment scheme, and packet size. In the following sections, the impact of each of these parameters on protocol efficiency is explored. For this purpose, some simple mathematical models based on probability and queuing theory prove helpful. This paper discusses some of the results derived from these models, while Reference 5 presents a more complete treatment of the subject including full derivations of the results presented here.

RETRANSMISSION

As noted above, the primary purpose of retransmission is to overcome loss or damage of data packets (or acknowledgments) by the communication network. The protocol parameter controlling retransmission is the *retransmission interval,* R. If no acknowledgment for a transmitted packet is received within time R, the packet will be retransmitted.

The choice of R can have significant effects on both throughput and delay. A small R minimizes mean delay since lost packets are retransmitted promptly. If there is significant variation in network transmission delay for a packet, then quick retransmission may reduce mean delay even when no packets are lost or damaged. Larger R, on the other hand, tends to maximize throughput since no bandwidth is "wasted" on unnecessary retransmissions. Assuming packets need not be delivered in order, there is no penalty for waiting until loss of a packet is certain before retransmitting it, since other packets can continue to be delivered.

To quantify these general observations, the following simple model proves useful. Let the network transmission delay be represented by a probability density function f(t) to allow for variation in delay. Both propagation time through the network (Tprop) and transmission time into the network (for a packet of length P and Host-to-network bandwidth of B, this is P/B) are included in f(t). f(t) typically has a high peak around the "normal" delay, and a small but long tail representing the possibility of occasional long delays (see Figure 1). If the probability of lost or damaged packets is LS, then f(t) includes an impulse at t=infinity with value LS (the probability that a packet never arrives).

f(t) and its associated cumulative distribution F(t) now represent the behavior of a packet of length P transmitted through the network. Assuming packets are retransmitted at intervals R, and that f(t) is identical for each (re)transmission of a packet (independence of successive transmissions),[12,13] the cumulative delay distribution until the first *successful* receipt of a packet, G(t), may be derived from F(t) and R.

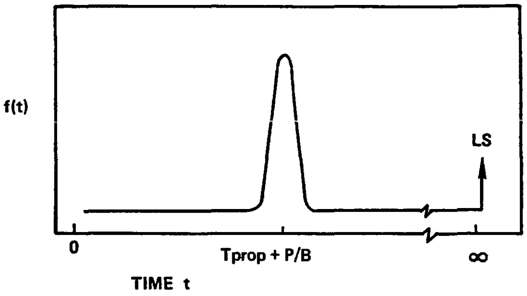


Figure 1—Network transmission delay density function f(t) including packet loss probability LS

$$\begin{aligned} G(t) &= \text{Prob}\{\text{at least one successful delivery by time } t\} \\ &= 1-\text{Prob}\{\text{no success by time } t\} \\ &= 1-\prod_{i=0}^{n-1}[1-F(t-i\cdot R)] \qquad n=\lceil t/R \rceil \end{aligned}$$

If f(t) represents roundtrip delay (time from transmitting a packet until an acknowledgment is received), then G(t) provides sufficient information to calculate the mean delay DL until the first successful acknowledged transmission, and the mean number of transmissions N. Since each successful transmission requires on the average N actual transmissions, throughput attainable will be proportional to a throughput factor TPretrans equal to the inverse of N.

To demonstrate this analysis, let f(t) be the Erlangian family of distributions:

$$f(t)=(k\cdot u)\cdot\frac{(k\cdot u\cdot t)^{k-1}}{(k-1)!}\cdot e^{-k\cdot u\cdot t}$$

With an appropriate choice of the shape parameter k (e.g., k=16), this distribution provides an adequate model of typical network transmission delays[12] (see Figure 2). Applying the above analysis to the Erlang-

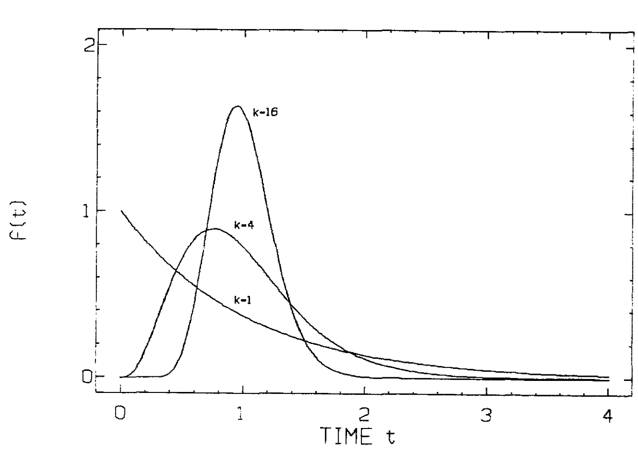


Figure 2—Erlangian probability density function, f(t), with mean=1 and shape parameter k=1,4, 16

ian f(t) with k=16 gives Figure 3, a plot of throughput versus delay for varying retransmission interval R and several packet loss probabilities LS (the results for exponential f(t) are shown dotted for comparison). The definite "knee" in the curves for nonzero LS indicate an optimal value of R. For larger R, delay is increased with little savings in throughput. For smaller, R, throughput is reduced (due to excessive retransmission) at little improvement in delay. This optimal value of R occurs at the time in the f(t) curve by which "most" transmissions would have succeeded (if no packets were lost or damaged). The optimal R is more clearly defined for f(t) with sharp peaks (more constant delay) and for higher packet loss probabilities.

To extend this analysis to protocols which perform sequencing, the assumption of independent identically distributed roundtrip delays for successive transmissions must be modified. In particular, loss of a packet or its acknowledgment causes acknowledgment of all subsequent packets to be delayed until the earlier error is corrected, even though other packets are successfully received. In this case, negative acknowledgments may improve performance by forcing prompt retransmission of the damaged packet, and suppressing retransmission of other outstanding packets (c.f. next section). Although negative acknowledgments help reduce performance losses, higher delay and lower throughput must be expected with a sequencing protocol if packets arrive significantly out of order.

FLOW CONTROL

Roundtrip delay is typically an order of magnitude greater than Host-to-network packet transmission time over a packet switching network. In this environment, a simple protocol that waits for acknowledgment of each packet before transmitting the next packet will be idle a large fraction of the time. Therefore, to achieve higher throughput, the sender must be allowed to transmit multiple packets before receiving any acknowledgments. Since each outstanding packet requires buffering and other communication resources, the amount of advance transmission required to achieve maximum throughput becomes an important efficiency question.

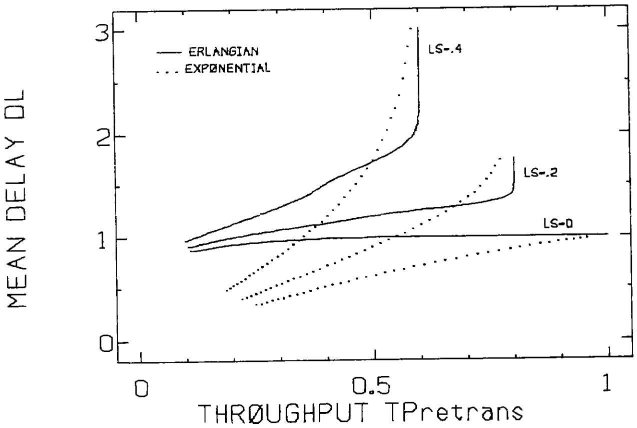


Figure 3—Mean delay DL vs. throughput factor TPretrans for Erlangian network transmission delay with mean=1 and k=16

On the other hand, it is also necessary to limit the flow of information from source process to destination process. The communication network itself must guard against internal congestion by imposing "congestion control" constraints on traffic sources.[2,14] More relevant to this study, each sender's transmission rate should be matched to the receiver's acceptance rate to minimize protocol resources required to support the given traffic. Achieving this matching is the primary purpose of a protocol's *flow control* mechanisms.

Most flow control strategies can be described in terms of a limited *window size* of Nwin packets (and/or bits) that may be transmitted but not yet acknowledged.[1,15-18] When this limit is reached, the sender must stop transmitting new packets (although retransmissions of pending packets may proceed) until more transmission "credits" are returned by the receiver (often associated with acknowledgments).

This situation can be modeled as a dual-server closed queuing system with the transmitter as one server, and the network and receiver as the other. (This second server is an "infinite" server since packets can proceed in parallel through the network.) Service time at the transmitter (Tlocal) is the Host-to-network transmission time for a packet (P/B), while service time 2 (Tnet) which includes propagation of the packet through the network, receiver processing, and return of an acknowledgment, is the roundtrip delay less service time 1. Define RHO as the ratio of service time 1 to service time 2. The window size Nwin defines the number of customers (packets) in the system (see Figure 4).

The utilization of server 1 (UT) is the fraction of time that the transmitter is active (allowed to transmit) with a given window size, and hence provides a good indication of throughput attainable. Figure 5 shows this utilization as a function of window size for several values of RHO and assuming exponentially distributed service times (high variance). Realistic values of RHO in a typical PSN are typically about 0.1 since roundtrip delays are an order of magnitude larger than Host-to-network transmission times. Throughput rises linearly with window size (note the logarithmic scale for window size) up to the half-way point, and then somewhat more slowly. For constant service times (no variance), throughput would rise linearly with window size all the way to its maximum value, while service time distributions (roundtrip delays) with intermediate variance would behave between these limits.

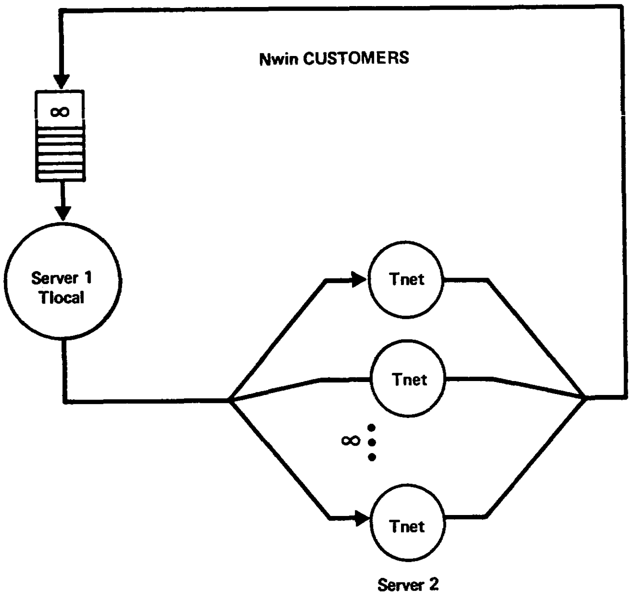


Figure 4—Queuing model of flow control

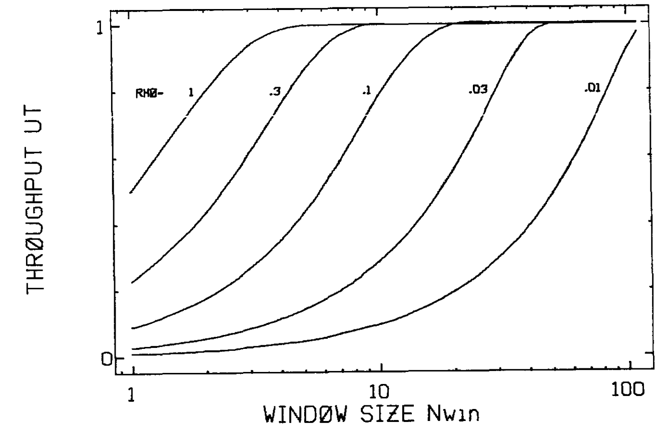


Figure 5—Throughput factor UT vs. flow control window size Nwin for various RHO-Tlocal/Tnet (the ratio of packet transmission time to roundtrip delay through the network)

This model provides information for both flow control concerns expressed above. To achieve *maximum* throughput, a window size approximately equal to the roundtrip delay divided by the Host-to-network transmission time for a packet is optimal (a larger window does not increase throughput much). This confirms the intuitive approach of "keeping the pipe full" between sender and receiver. To *limit* throughput to a desired fraction of the bandwidth available, the same fraction of this optimal window size should be used.

As long as packet loss and damage probabilities remain low, the above analysis shows how throughput may be *flow control limited* by a small window size. In networks where transmission errors are more likely, throughput may become *retransmission limited* because retransmissions of pending packets have priority over new transmissions. In general, achievable throughput will be the minimum due to either flow control or retransmission constraints.

A third similar constraint may be imposed by the *source* further restricting the window size for a connection (below what the receiver might allow) in order to share its communication resources fairly among all connections. For example, the ARPANET TIP[19] limits each terminal (up to 300 baud) to a window of 6-12 characters in order to share its relatively scarce buffer space among all users. In the next section, the effects of destination buffer space limits or protocol performance will be explored.

DESTINATION BUFFERING AND ACKNOWLEDGMENT STRATEGIES

Buffering of received packets at the destination is necessary to smooth uneven production and consumption rates (hold packets until they can be processed) and to hold out-of-order arrivals for proper sequencing (if the protocol provides a sequencing function). Inadequate space for either purpose means that successfully received packets may have to be discarded if no space is available to hold them, increasing retransmission rate and delay.

In an attempt to avoid these problems, "conservative" acknowledgment strategies delay returning permission to send new data (c.f. last section) until buffer space to receive it has actually been made available by "consuming" arrived packets or furnishing additional space. In this case, no arriving packets need be discarded, but roundtrip delay for flow control credits is increased because processing time is included, and hence throughput may be reduced as shown in the last section.

Under favorable conditions, new flow control credits can be returned immediately on successful receipt of a packet, reducing the roundtrip time. As long as packet arrivals and receiver processing proceed smoothly (at regular intervals), no packets will be discarded. A larger window size than the buffer space available may be used to achieve higher throughput with reduced buffering requirements. Storage space along the network transmission path actually provides the rest of the space in the window.

Another simple queuing model provides insight into the performance of this optimistic strategy. The receiver is represented by a single server with mean service (processing) rate u, and size-limited queue of Nbuf packet buffers. Packets arrive at a mean rate A determined by retransmission and flow control constraints discussed earlier, and the ratio of arrival rate to processing rate is RHO = A/u. Assuming exponential distributions, well-known queuing results[20] give the probability that an arriving packet will find all the buffers full, and have to be discarded.

$$\text{Pfull}(\text{Nbuf},\text{RHO}) = (1-\text{RHO}) \cdot \frac{\text{RHO}^{\text{Nbuf}}}{1-\text{RHO}^{\text{Nbuf}+1}}$$

Three general cases may be distinguished.

For RHO>>1, (arrival rate>>processing rate), nearly all arriving packets will be discarded regardless of the size of Nbuf. In this case, throughput is clearly *receiver rate limited,* and the window size should be reduced to limit the sender's transmission rate to the receiver's processing rate.

For RHO<<1, very few packets will be discarded even if Nbuf is small. A fast receiver requires very little buffering, and buffer pooling among multiple connections may be advantageous.

For RHO = 1, (matched sending and receiving rates), the number of buffers required depends heavily on the "smoothness" of source and destination activity. Very regular activity allows a minimum of buffering. Unfortunately, irregular packet arrival and processing rates are more typical in computer networks, particularly in multiprogramming systems where process activity occurs in bursts. If process scheduling intervals in multiprogramming systems are large compared to roundtrip delays, then large window sizes and buffer allocations may be necessary to achieve high throughput.

PACKET SIZE

In transmitting large amounts of information between processes, a communication protocol must determine the amount of data to transmit in each packet. The primary result of varying packet size is to vary transmission delay through the network, f(t). Transmission delay in a store-and-forward network has a large component proportional to packet length because the transmission time on each hop between switching nodes is equal to packet length divided by bandwidth[7] (although some networks either fragment or combine submitted packets before internal transmission).

Assuming this proportionality holds, shorter packets mean lower per packet delay, with ensuing effects on protocol performance as described in earlier sections. Unfortunately, overhead for short packets increases since each packet carries a fixed amount of header and control information, and more acknowledgments and general processing for the same amount of data will be necessary.[4] Hence, maximum throughput attainable decreases while line efficiency (and total cost in bits transmitted) increases for shorter packets. Longer packets reduce overhead and allow higher throughput at the cost of increased delay. Packet switching networks typically impose an upper bound on the size of packets submitted for transmission in order to manage their internal resources and to ensure prompt access to other customers.[2]

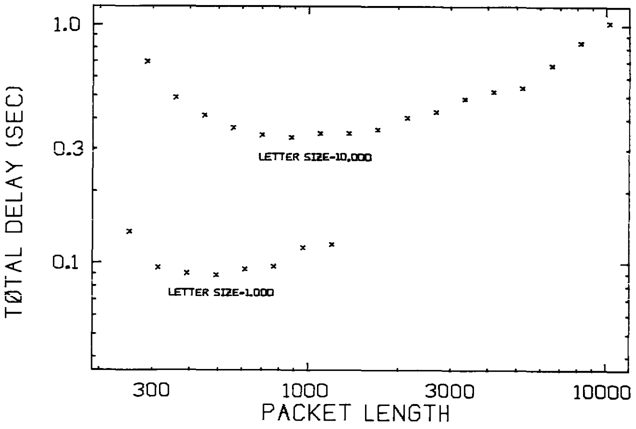


Figure 6—Total delay vs. packet length for two block (letter) sizes

Network bandwidth=50 kb/sec
Number of hops=5
Header length=200 bits
Packet less probability=0

To illustrate some of these trade-offs, Figure 6 shows the total time to transmit a large block of information using different packet sizes, assuming typical PSN characteristics (but no packet loss) and transmission delay proportional to packet length. Using the smallest packets, throughput is so low that total delay is high. For very large packets, the increasing delay per packet also leads to high total delay. The "optimal" value for this application results from an intermediate packet size.

Other applications may have other priorities for cost, throughput, and delay performance.[9] Transaction or interactive applications may select short packets to achieve low delay at somewhat higher cost and reduced throughout. Minimum cost or maximum throughput users willing to tolerate larger average delays may use long packets. "Real-time" traffic requiring moderate delay and good throughput for moderate block lengths may use intermediate packet sizes. As network bandwidth increases and error rates drop, the impact of packet length on protocol performance should lessen.

CONCLUSIONS

Distributed processing environments such as computer networks demand new techniques for providing reliable and efficient communication between remote processes. Good performance of communication protocols developed for this purpose depends heavily on tuning protocol parameters to complement network transmission characteristics. This paper has discussed the impact on communication efficiency of major protocol policies concerning retransmission, flow control, buffer allocation, acknowledgment, and packet size.

More complete resnlts and more detailed development of the models underlying these results are contained in another paper by the author.[5] Experiments are currently under way to verify the predictions of these models using the newly developed Transmission Control Program[17,21,22] as an interprocess communication protocol.

ACKNOWLEDGMENTS

The author wishes to thank V. Cerf, Y. Dalal, R. Metcalfe, and S. Gaines, for their helpful discussion of the issues and review of the manuscript. Personnel and facilities at several ARPANET sites were also instrumental in supporting this work.

REFERENCES

1. Pouzin, L., *Basic Elements of a Network Data Link Control Procedure (NDLC)*, INWG Note 54, NIC 30375, January 1974. Also in ACM *Computer Communication Review 5*, 1, January 1975, pp. 6-23.
2. Crowther, W. and others, "Issues in Packet Switching Network Design," *Proc. National Computer Conf.*, 1975, AFIPS Press, pp. 161-175.
3. Sunshine, C. A., *Interprocess Communication Protocols for Computer Networks*, Stanford University DSL Technical Report #105, December 1975. (Ph.D. Thesis).
4. Kleinrock, L., W. E. Naylor and H. Opderbeck, *A Study of Line Overhead in the ARPANET*, INWG Note #71, September 1974. Also in *Comm. ACM 19*, 1, January 1976, pp. 3-13.
5. Sunshine, C. A., *Efficiency of Interprocess Communication Protocols for Computer Networks*, The Rand Corp. P-5614, March 1976.
6. Burton, H. O. and D. D. Sullivan, "Errors and Error Control," *Proc. IEEE 60*, 11, November 1972, pp. 1293-1300.
7. McQuillan, J. M., W. R. Crowther, B. P. Cosell, D. C. Walden and F. E. Heart, "Improvements in the Design and Performance of the ARPA Network," *Proc. Fall Joint Computer Conf.*, 1972, AFIPS Press, pp. 741-754.
8. Pouzin, L., *Efficiency of Full-Duplex Synchronous Data Link Procedures*, INWG Note #35, June 1973. Also NIC 18255.
9. Opderbeck, H. and L. Kleinrock, "The Influence of Control Procedures on the Performance of Packet-Switched Networks," *Proc. National Telecommunications Conf.*, San Diego, December 1974. Also INWG Note #62, September 1974.
10. Metcalfe, R. M., *Packet Communication*, M.I.T. Project MAC Report TR-114, December 1973. (PhD Thesis, Harvard University).
11. Kleinrock, L. and W. E. Naylor, "On Measured Behavior of the ARPA Network," *Proc. National Computer Conf.*, 1974, AFIPS Press, pp. 767-780.
12. Forgie, J. W. and C. K. McElwain, *ARPANET Delay Measurements*, NSC Note 70, M.I.T. Lincoln Labs, July 1975.
13. Naylor, W. E., personal communication.
14. Kahn, R. E. and W. R. Crowther, "Flow Control in a Resource-Sharing Computer Network," *IEEE Trans. on Communications*, COM-20, 3, June 1972, pp. 539-546.
15. Carr, S., S. Crocker and V. Cerf, "Host/Host Protocol in the ARPA Network," *Proc. Spring Joint Computer Conf.*, 1970, AFIPS Press, pp. 589-597.
16. Belsnes, D., *Flow Control in Packet Switching Networks*, Stanford University DSL Technical Note #50 and INWG Note #63, October 1974.
17. Cerf, V. G. and R. E. Kahn, "A Protocol for Packet Network Intercommunication," *IEEE Trans. on Communications*, COM-22, May 1974, pp. 637-648.
18. Zimmerman, H., "The CYCLADES End-End Protocol," *Proc. Fourth Data Communications Symp.*, Quebec City, Canada, October 1975, IEEE 75 CH1001-7 DATA, pp. 7-21 to 7-26.
19. Ornstein, S. M. and others, "The Terminal IMP for the ARPA Computer Network," *Proc. Spring Joint Computer Conf.*, 1972, AFIPS Press, pp. 243-254.
20. Kleinrock, L., *Queueing Systems*, Vol. I, John Wiley, New York, 1975.
21. Cerf, V. G., Y. Dalal and C. Sunshine, *Specification of Internet Transmission Control Program*, INWG Note 72, revised December 1974.
22. Cerf, V. G., *ARPA Internetwork Protocols Project Status Report*, Technical Note 68, Digital Systems Lab, Stanford University, November 1975.
23. Roberts, L. G. and B. Wessler, "The ARPA Computer Network," in *Computer Communication Networks*, Abramson and Kuo, editors, Prentice Hall, 1972.

Performance of file directory systems for data bases in star and distributed networks*

by WESLEY W. CHU
University of California, Los Angeles
Los Angeles, California

ABSTRACT

Three classes of file directories for distributed data bases are studied: The centralized directory system, the local directory system, and the distributed directory system. The parameters considered are communication cost, storage cost, code translation cost, query rate, update rate, directory size, and directory response time. This study reports the cost performance tradeoffs of these three classes of directory systems in star and distributed networks and provides a guide in the design of file directory systems when operating in these network environments.

INTRODUCTION

In the automation of large information systems, a major portion of the planning is concerned with methods of storing, updating, retrieving, and distributing large quantities of information in an information processing system. Examples of such efforts are found in business, medicine, library and management information systems. These systems, which may consist of several geographically separate divisions, need to process information files in common and thus form a network information processing system known as a distributed data base. One of the significant advantages of this network environment is the resource sharing capability. Problems that arise in the design of distributed data bases include file allocation policy,[1,2,3] avoidance of file deadlock,[4] file directory design, file partitioning policy, file reliability, privacy, and security issues, and interfacing considerations. This paper concerns itself with the allocation of directories in a distributed data base.

A directory is a listing of files available to the users of the network. Such a directory will enable a user at any node to determine where in a network a specific sharable file exists. One can consider such a directory to be similar to a card catalog of a public library. These sharable files are referred to as public files. Users at each node may offer to list their files in this directory of public files for sharing purposes. A user may interrogate this list to determine its contents or to obtain information on where a specific sharable file exists. The list of non-shared files is assumed to be stored at the computer that is known to its users and, therefore, is not considered here. Further, we assume that each computer has its own local directory which consists of all the public files stored in that computer. To locate a file that is not in the local computer, the user must consult the file directory.

There are several ways to design the file directory: centralized file directory, local file directory and distributed file directory. Based on the computer network topology, operating cost (communication cost, storage cost, and code translation cost), directory query rate and directory update rate, we use mathematical models to study the operating cost and response time of these directory systems as a function of directory query rate, directory update rate, and the ratio of storage cost to communication cost. Two numerical examples are given (one with a star network topology, the other with a distributed network topology) to illustrate the applications of the models. The results provide us with insights on how to optimally plan for a file directory for a distributed data base.

THE MODEL

In this section, several mathematical models are introduced to study the performance of file directory systems for operating in the star network and distributed network topologies. Let us first consider the centralized file directory system.

Centralized file directory system

Single master directory

In the centralized file directory system, a master directory is located at one of the computers. When a

* This research was supported by the U.S. Office of Naval Research, Contract No. N00014-75-C-0650.

user requires a file that is not stored at his local directory, he consults the master directory to find out the location or content of that file. The centralized directory must be updated when there is a change in the storage location or contents of a file, or a creation of a new version of the file. Directory updating in such a system is relatively easy. However, there are communication costs incurred for each transaction. The operating cost for such a system per unit time is

$$C_c=\underbrace{\sum_{\substack{i=1\\i\neq d}}^{n} C_{id}{}^t s_{id} l_i \sum_{j=1}^{m_i} q_{ij}(2+p_{ij})}_{\text{communication cost}}+\underbrace{\sum_{\substack{i=1\\i\neq d}}^{n}\sum_{j=1}^{m_i}[T_i q_{ij}(2+P_{ij})]}_{\text{translation cost}}+\underbrace{C_d{}^s F}_{\text{storage cost}} \tag{1}$$

Where: m_i = total number of files in the ith computer

n = total number of computers in the network

d = location of the master directory

$C_{id}{}^t$ = transmission cost per character and per unit distance from the ith computer to the master directory

s_{id} = distance between the ith computer to the master directory.

l_i = average record length in characters for each transaction at the ith computer, $i=1, 2, \ldots, n$

q_{ij} = directory query rate by the jth file at the ith computer per unit time. $j=1, 2, \ldots, m$.

P_{ij} = normalized directory update rate (the ratio of directory update rate to directory query rate) of the jth file generated by the ith computer. $0\leq P_{ij}\leq 1$.

T_i = code translation cost per transaction by the ith computer at the master directory.

$C_d{}^s$ = storage cost per character per unit time at the master directory

F = size of the master file directory in characters.

The first term in Equation (1) represents the transmission cost for querying and updating the file directory, the second term represents the code translation cost (due to non-uniform information representations) for transactions (querying and updating) performed by all the computers at the directory, and the last term is the cost for storing the master file directory.

To simplify the notation in Equation (1), we let $q_i=\sum_{j=1}^{m_i} q_{ij}$. The q_i is the total number of queries generated per unit time by the ith computer for the directory. Further, we assume that all the files at the ith computer have the same directory update; that is, $P_{ij}=P_i$ for all j. Equation (1) then reduces to

$$C_c=\sum_{\substack{i=1\\i\neq d}}^{n} C_{id}{}^t s_{id} l_i q_i (2+p_i)+\sum_{\substack{i=1\\i\neq d}}^{n} T_i q_i (2+P_i)+C_d{}^s F \tag{2}$$

Extended centralized directory

In a centralized file directory, once the user finds the location or description of a file, he can append this information onto his local directory. Should the user use this file again, the directory information for this file can be obtained from his local directory, thereby reducing the communication cost as well as time for querying the master directory. However, when the information of that file at the master directory is updated, we also require updating of the information of that file in the local directories. Therefore, for notification of future updates in the master directory, the list of local directories that have appended file information is recorded in the master directory. The operating cost of such a system is:

$$C_{EC}=\sum_{\substack{i=1\\i\neq d}}^{n} C_{id}{}^t s_{id} l_i q_i p_i+\sum_{\substack{i=1\\i\neq d}}^{n} q_i p_i T_i + C_d{}^s(F+F_e)$$
$$+\sum_{\substack{i=1\\i\neq d}}^{n}\alpha_i C_i{}^s F+\sum_{\substack{i\neq k\\i=1}}^{n}\sum_{\substack{k\neq d\\k=1}}^{n} q_i p_i \beta_{ki}(C_{dk}{}^t s_{dk} l_k + T_{dk}) \tag{3}$$

where α_i = fraction of the master directory being appended at the local directory of the ith computer, $0\leq\alpha_i\leq 1$ $i=1, 2, \ldots, n$.

β_{ki} = probability that a file at the kth computer has a transaction at the ith computer, $0\leq\beta_{ki}\leq 1$, $i=1, 2, \ldots, n$ and $k=1, 2, \ldots, n$.

T_{dk} = translation cost for the master directory to perform a transaction with the kth computer.

F_e = the size of the list of all the files stored at the master directory that requires notification for directory updates (in characters).

We assume that compared with the overall operating cost, the first cost of locating a file in the master directory is negligible. The first term in Equation (3) is the transmission cost for updating the master directory, the second term is the code translation cost for updating the master directory, the third term is the cost for storing the directory and the list of the transaction records for notification of future file directory updates, the fourth term is the cost for storing the extended file directories at all the local directories, and the last term is the communication cost and translation cost for the master directory to update the relevant appended local directories.

Multiple master directories

When the computers in a network are clustered in groups, it is often cost effective to provide a master directory at each of these clusters. The savings in communication cost for a multiple master directory system could far outweigh the cost of storing and updating the file directories. We can partition the n computers in the system into r clusters ($r \leq n$) such that

$$\sum_{j=1}^{r} n_j = n$$

where n_j is the number of computers in the jth cluster. During normal operation, the computers in the jth cluster will query the jth directory which is a member of the jth cluster. One way to partition the n computers into r groups is to base on the network topology such that the partitioned clusters yield minimum communication costs. Another way is to base the clusters on directory query rate and directory update rate to achieve minimum response time. The operating cost for such a system is:

$$C_M = \sum_{k=1}^{r} \left\{ \sum_{\substack{\{i\}\epsilon d_k \\ d_k \neq i}} C_{id_k}{}^t s_{id_k} l_i q_i (2+p_i) + \sum_{\substack{\{i\}\epsilon d_k \\ i \neq d_k}} T_i q_i (2+p_i) \right\}$$
$$+ \sum_{k=1}^{r} C_{d_k}{}^s F \quad \sum_{g=1}^{r} \sum_{\substack{k=1 \\ k\neq g}}^{r} \sum_{\{i\}\epsilon d_k} (C_{d_k d_g}{}^t \cdot s_{d_k d_g} \cdot l_i + T_{d_k d_g}) p_i q_i \qquad (4)$$

where r = number of master directories in the system

d_k = the kth master directory, $k = 1, 2, \ldots, r$.

$\{i\}\epsilon d_k$ = the set of computers that uses the kth master directory

$C_{d_k}{}^s$ = storage cost per character per unit time of the kth master directory

$C_{id_k}{}^t$ = transmission cost per character per unit distance from the ith computer to the kth master directory, $i = 1, 2, \ldots, n$ and $k = 1, 2, \ldots, r$.

The last term of (4) is the communication cost and translation cost for updating the rest of the multiple directories in the system. For the single master directory case ($r=1$), Equation (4) reduces to (2).

Local file directory system

In the local directory case, there is no master directory in the system. When a requested file is not stored in the user's local directory, the user queries all the other local directories in the system until the requested file has been located. Such a directory system requires high communication cost and translation cost, as well as search time for locating the file. For a system of n computers, it requires an average of (n-1)/2 directory queries to locate a file. Assuming the directory can only be updated by its owner, updating is done at its local directory which does not require communications cost. The operating cost for such a system is

$$C_L = \tfrac{1}{2}\left[\underbrace{2 \sum_{i=1}^{n} \sum_{\substack{k=1 \\ i\neq k}}^{n} C_{ik}{}^t s_{ik} l_i \sum_{j=1}^{m_i} q_{ij}}_{\text{Communication Cost}} + \underbrace{2 \sum_{i=1}^{n} \sum_{\substack{k=1 \\ i\neq k}}^{n} T_{ik} \sum_{j=1}^{m_i} q_{ij}}_{\text{Translation Cost}} \right] \qquad (5)$$

where T_{ik} is the code translation cost per transaction for the ith computer at the kth computer. The first term of Equation (5) is the expected communication costs for querying to and replying from all the local directories for those files that are not stored in the local computer, and the second term is the expected code translation costs for transactions with all the local directories in the system. The factor 1/2 is for taking the average operating cost. Since there is no master directory in such a system, there is no storage cost in Equation (5).

For simplicity in notation, we let $q_i = \sum_{j=1}^{m_i} q_{ij}$. Eq. (5) becomes

$$C_L = \sum_{i=1}^{n} \sum_{\substack{k=1 \\ k\neq 1}}^{n} [C_i{}^t s_{ik} l_i + T_{ik}]\, q_i \qquad (6)$$

Let us consider the case where each of the computers contains a routing table which routes the directory query directly to the other computers rather than returning the negative query reply to the sender. As a result, the expected total communication cost can be greatly reduced, particularly if the routing sequence takes into consideration the probability of finding the file in the directory. The total operating cost reduces to

$$C_L' = \gamma \sum_{i=1}^{n} \sum_{\substack{k=1 \\ i\neq k}}^{n} [C_{i_k}{}^t s_{ik} l_i q_i + T_{ik} q_i] + \sum_{i=1}^{n} C_{ik}{}^t s_{ik} l_i q_i \qquad (7)$$

where $0 < \gamma < 0.5$. The value of γ depends on the network topology and the policy used in the routing table. The last term of Eq. (7) is the communication cost for replying to the queries.

Distributed file directory system

In the distributed directory case, each computer in the system has a master directory. The advantage of this system is its fast response time. The disadvantage of this system is the cost of storing master file directories at each computer as well as the communication cost for updating all these directories. The operating cost per unit time of such a system is

$$C_D = \underbrace{\sum_{i=1}^{n} \sum_{\substack{k=1 \\ i\neq k}}^{n} C_{ik}{}^t s_{ik} l_i \sum_{j=1}^{m_i} q_{ij} p_{ij}}_{\text{Communication Cost}}$$
$$+ \underbrace{\sum_{i=1}^{n} \sum_{\substack{k=1 \\ i\neq k}}^{n} T_{ik} \sum_{j=1}^{m_i} q_{ij} p_{ij}}_{\text{Translation Cost}} + \underbrace{\sum_{i=1}^{n} C_i{}^s F}_{\text{Storage Cost}} \qquad (8)$$

The first term of Equation (8) is the communication cost for updating all the distributed master directories in the system, the second term is the translation cost associated with the updating of all the distributed directories, and the last term is the cost for storing all the master directories. If we assume that the directory update rates of all the files at the ith computer are identical, then $P_{ij}=P_i$. Again for simplicity in notation, we let $q_i=\sum_{j=1}^{m_i} q_{ij}$. Then (8) reduces to

$$C_D=\sum_{\substack{i=1\\ i\neq k}}^{n}\sum_{k=1}^{n}(C_{ik}{}^t s_{ik} l_i+T_{ik})\,p_i q_i+\sum_{i=1}^{n}C_i{}^s F \qquad (9)$$

OPERATING COST TRADEOFFS OF DIRECTORY SYSTEMS

The intersection of two operating cost curves $C_x(P)$ and $C_y(P)$ (Figures 1 and 2) represent the cost tradeoff point (in terms of update rate) for directory systems x and y. If we assume that all the computers in the system have identical directory update rates, then the operating cost is a linear function of the normalized update rate P. Thus $C_x(P)$ and $C_y(P)$ can be expressed as:

$$C_x(P)=a_xP+b_x$$

$$\text{and}\quad C_y(P)=a_yP+b_y$$

where a_x and a_y are incremental costs for directory updates and b_x and b_y are fixed directory operating costs. The intersection point of $C_x(p)$ and $C_y(P)$, $P(x,y)$, satisfies

$$P(x,y)=\frac{b_y-b_x}{a_x-a_y} \qquad (10)$$

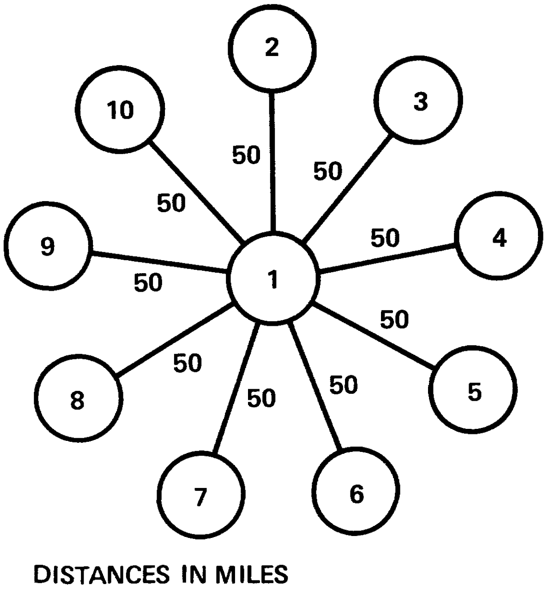
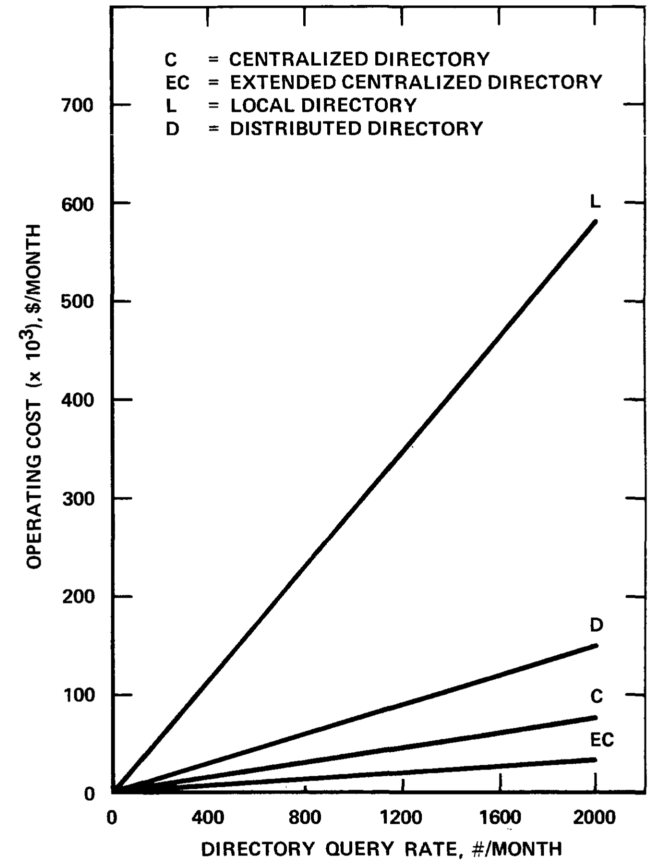


Figure 1—Performance of directory systems for a star network $C^t/C^s=10$ month/mile—(a) A star network



(b) Directory operating cost vs. directory query rate

Let us now consider the intersection of the cost curves for the centralized and extended centralized directory systems. From Equation (2), we have

$$a_c=\sum_{\substack{i=1\\ i\neq d}}^{n}C_{id}{}^t s_{id} l_i q_i+\sum_{\substack{i=1\\ i\neq d}}^{n}T_i q_i$$

$$b_c=\sum_{\substack{i=1\\ i\neq d}}^{n}2C_{id}{}^t s_{id} l_i q_i+\sum_{\substack{i=1\\ i\neq d}}^{n}2T_i q_i+C_d{}^s F$$

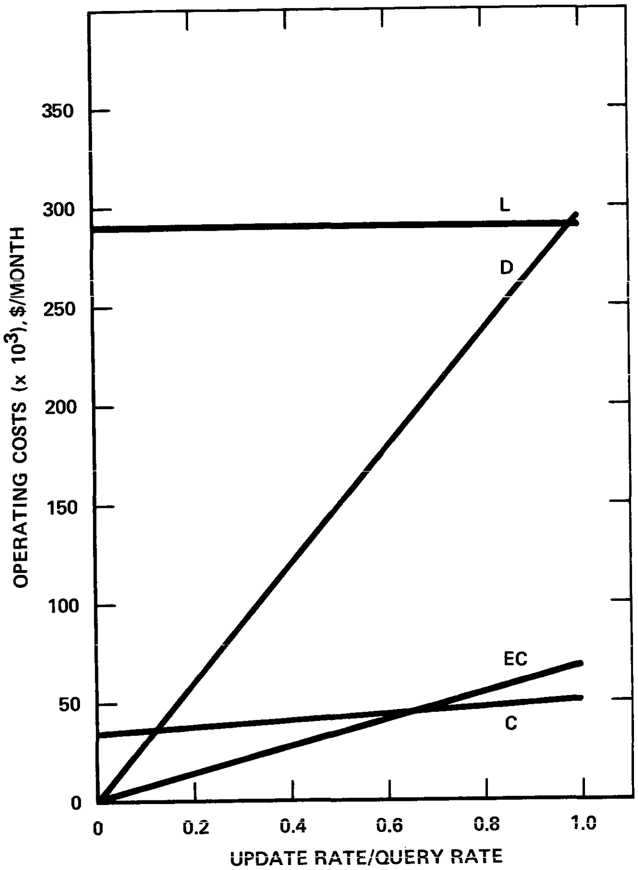
From (3), we have

$$a_{EC}=\sum_{\substack{i=1\\ i\neq d}}^{n}C_{id}{}^t s_{id} l_i q_i+\sum_{\substack{i=1\\ i\neq d}}^{n}q_i T_i+\sum_{\substack{i\neq k\\ i=1}}^{n}\sum_{\substack{k\neq d\\ k=1}}^{n}q_i\beta_{ki}(C_{dk}{}^t s_{dk} l_k+T_{dk})$$

$$b_{EC}=C_d{}^s(F+F_e)+\sum_{\substack{i=1\\ i\neq d}}^{n}\alpha_i C_i{}^s F$$

Substituting a_c, a_{EC}, b_c, and b_{EC} into (10) and simplifying, we have

$$P(c,EC)=\frac{C_d{}^s F_e-\sum_{\substack{i=1\\ i\neq d}}^{n}\alpha_i C_i{}^s F+\sum_{\substack{i=1\\ i\neq d}}^{n}2T_i q_i+\sum_{\substack{i=1\\ i\neq d}}^{n}2C_{id}{}^t s_{id} l_i q_i}{\sum_{\substack{i\neq k\\ i=1}}^{n}\sum_{k\neq d}^{n}q_i\beta_{ki}(C_{dk}{}^t s_{dk} l_k+T_{dk})} \qquad (11)$$



(c) Directory operating cost vs. normalized directory update rate. Query rate=1000/month

In order to simplify equation (11), we assume that (1) the communication cost is much higher than the storage cost so that the storage cost becomes negligible, (2) all the computers in the system are using the same software code, thus translation cost is not required; that is, $T_i=0$, and (3) $\beta_{ki}=\beta$, $l_i=l$, and $q_i=q$ for all i, then (11) reduces to

$$P(c,EC) \doteq \frac{2}{(n-1)\cdot\beta} \qquad (12)$$

For example, if $n=10$ and $\beta=1/3$, then $P(c,EC)=2/3=0.667$. Thus for a network with ten computers operated in the above stated environment, when the directory update rate of each computer is less than 67 percent of its query rate, the extended centralized directory system yields a lower operating cost than that of the centralized directory system.

We shall now consider the directory operating cost tradeoff between the centralized directory system and the distributed directory system. From (9), we have

$$a_D=\sum_{i=1}^{n}\sum_{\substack{k=1\\ i\neq k}}^{n}(C_{ik}{}^t s_{ik} l_i+T_{ik})\,q_i$$

$$b_D=\sum_{i=1}^{n} C_i{}^s F$$

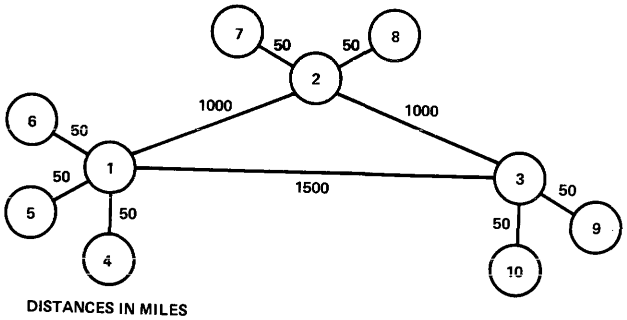
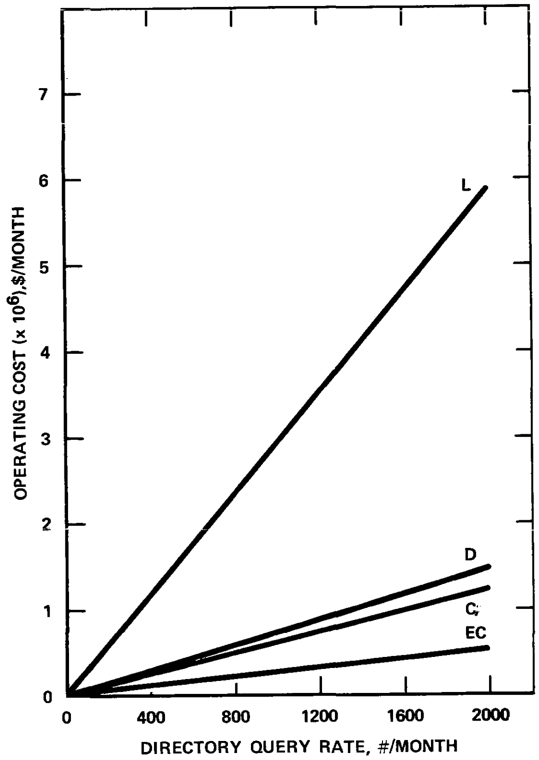


Figure 2—Performance of directory system for a distributed network. $C^t/C^s=10$ month/mile—(a) A distributed network

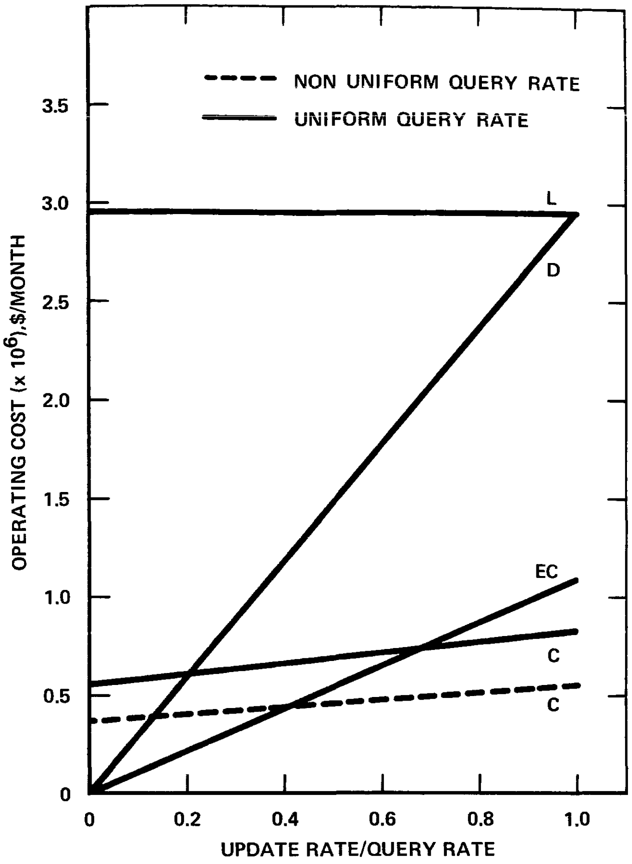
Substituting a_c, a_D, b_c, and b_D into (10), we have

$$P(c,D)=\frac{-\sum\limits_{\substack{i=1\\ i\neq d}}^{n} C_i{}^s F+\sum\limits_{\substack{i=1\\ i\neq d}}^{n} 2C_{id}{}^t s_{id} l_i q_i+\sum\limits_{\substack{i=1\\ i\neq d}}^{n} 2T_i q_i}{\sum\limits_{\substack{i=1\\ i\neq k}}^{n}\sum\limits_{\substack{k=1\\ i\neq d}}^{n}(C_{ik}{}^t s_{ik} l_i+T_{ik})\,q_i} \qquad (13)$$

Compared with the communication cost, the storage cost again can be assumed to be negligible. Further we



(b) Directory operating cost vs. directory query rate. Normalized directory update rate=0.25



(c) Effect of query rate on operating cost. Master directory at computer 1. Uniform query rate query case: $q_i=1000$ queries/month, $i=1, 2, \ldots, 10$; Non-uniform Query Case: $q_i=1500$ queries/month $i=1, 2, \ldots, 5$ and $q_i=500$ queries/month, $j=6, 7, \ldots, 10$

assume that $l_i=1$, $q_i=q$, for $i=1,2, \ldots, n$; $C_{ik}{}^t s_{ik} \doteq C_{id}{}^t s_{id}$ and $T_i=T_{ik}=0$. Then (13) reduces to

$$P(c,D) \doteq \frac{2}{(n-1)} \tag{14}$$

For a network with ten computers operated in the above stated environment, from (14) we know that $P(C,D) \doteq 0.22$ which implies that when the directory update rate is less than 22 percent of its query rate, the distributed file directory yields a lower operating cost than the centralized file directory.

We shall now consider the intersection of the local file directory cost curve $C_L(P)$ with the distributed file directory cost curve $C_D(P)$. From (6), we have

$$a_L = 0$$

$$b_L = \sum_{i=1}^{n} \sum_{\substack{k=1 \\ i \neq k}}^{n} [C_{ik}{}^t s_{ik} l_i q_i + T_{ik} q_i]$$

Substituting a_L, a_D, b_L and b_D into (10), we have

$$P(L,D) = \frac{-\sum_{i=1}^{n} C_i{}^s F + \sum_{i=1}^{n} \sum_{\substack{k=1 \\ i\neq k}}^{n} (C_{ik}{}^t s_{ik} l_i + T_{ik}) q_i}{\sum_{i=1}^{n} \sum_{\substack{k=1 \\ i\neq k}}^{n} (C_{ik}{}^t s_{ik} l_i + T_{ik}) q_i} \tag{15}$$

When $C_{ik}{}^t \gg C_i{}^s$, $P(L,D) \to 1$. This implies that when the communication cost is high as compared to the storage cost and when the directory update rate is less than the directory query rate, the distributed file directory system yields a lower operating cost than that of the local file directory system.

Comparing the approximate results obtained from Equations (12), (14), and (15) with those from direct computation as shown in Figure 3, we note that they agree quite well at high communication cost. Therefore (12), (14), and (15) may be used to estimate the approximate operating cost tradeoffs for high communication cost cases.

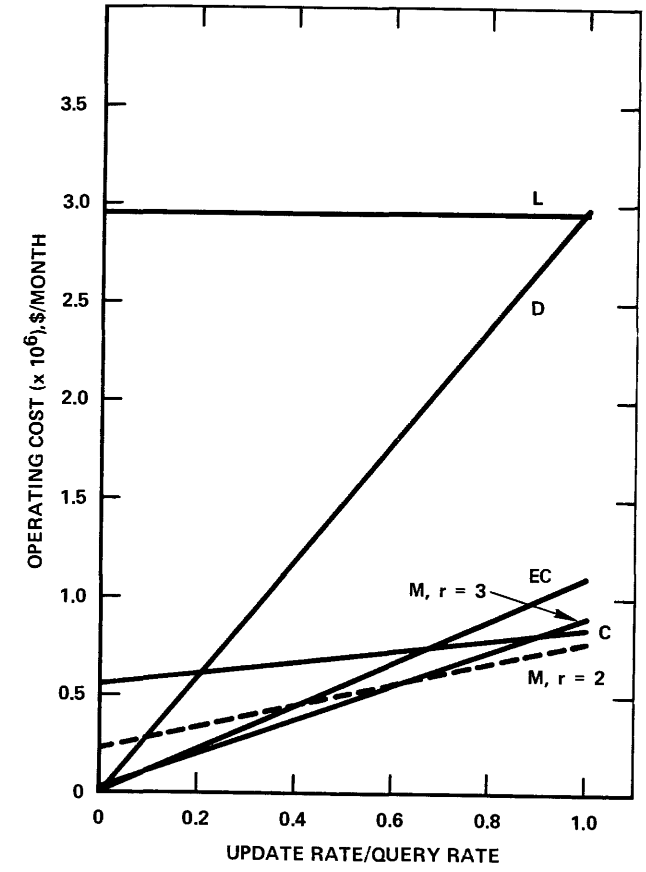


Figure 3—Performance of multiple directory systems for the distributed network. $C^t/C^s=10$ month/mile, and query rate$=$1000 queries/month. For $r=2$, master directories at computers 1 and 2; For $r=3$, master directories at computers 1, 2, and 3

DIRECTORY QUERY RESPONSE TIME

In this section, we shall consider the query response time for various directory systems. The expected response time for the i^{th} computer to query its directory is defined as from initiation of a query at the i^{th} computer to the directory until the start of its reception. The expected response time consists of the waiting time at the input queue of the directory for processing the query $t_1(i)$, the waiting time at the output queue of the directory for transmission $t_2(i)$, the time to transmit the query to and its reply from the directory $t_3(i)$, and the directory processing time $t_4(i)$. The processing time consists of code translation, searching, and accessing. It depends on the file structure of the directory as well as the access time of the storage device and could be different from one system to another and should be known to the users. Here, therefore, we only consider the delay incurred at the input queue and output queue(s) of the directory, and the time to transmit the query on the communication channel. Let us denote the sum of these components as $t_{i,d}$, known as the directory query response time from the i^{th} computer to the directory; thus

$$t_{i.d}=t_1(i)+t_2(i)+t_3(i) \tag{16}$$

Clearly, the real query response time incurred by the users, $t^r_{i.d}$, is equal to the sum of $t_{i.d}$ and $t_4(i)$.

The arrivals at the input queue of the directory are the queries and updates generated by all the computers. The arrivals at the output queue(s) of the directory are the query replies generated at the directory. We shall assume these query arrivals can be approximated by a Poisson Process. The time between interrupts by the computer to process directory queries is $1/\mu_1$. Since the communication line transmits at a constant rate of R characters/second, the time to transmit (send or reply) a query of l character is $1/\mu_2=l/R$. Under these conditions, the waiting time can be computed from known queuing theory results. The average waiting time for a single server queuing system with Poisson arrivals and constant service time,[5] M/D/1, is

$$W=\frac{1}{\mu}\cdot\frac{\lambda}{2(\mu-\lambda)}$$

where

$1/\mu$ = time to service a query in seconds
λ = average number of queries arrived/second

Assuming that queries and their replies generated at the directory input and output queues can be approximated as a Poisson process, then the query response time equals

$$t_{i,d}=t_1(i)+t_2(i)+t_3(i)=\frac{\lambda_1}{2\mu_1(\mu_1-\lambda_1)}+\frac{\lambda_2}{2\mu_2(\mu_2-\lambda_2)}+2/\mu_2 \tag{18}$$

Let us first consider the centralized directory system. The total number of queries arriving at the directory, λ_d, consists of directory queries and their updates from all the computers in the system. Thus $\lambda_1=\lambda_d=\sum_{i=1}^{n}q_i(1+p_i)$. Since the directory does not have to reply to update traffic, and since each destination has its own output queue, the arrival rate at the output queue for the i^{th} computer is equal to $\lambda_2=\lambda_{di}=q_i$. By substituting λ_1 and λ_2 into (18), we have

$$t_{i,d}=\frac{\sum_{i=1}^{n}q_i(1+p_i)}{2\mu_1\left[\mu_1-\sum_{i=1}^{n}q_i(1+p_i)\right]}+\frac{lq_i}{2R[R/l-{}_iq]}+\frac{2l}{R} \tag{19}$$

For the computer that stores the master directory, $t_2(d)=t_3(d)=0$. The expected query response time $t_{d.d}=t_1(d)$.

Let us now consider the extended centralized directory system. For those files that have not yet been queried at the directory by the i^{th} computer, the response time is similar to (19) except that the directory query rate is much smaller and the directory output queue for the i^{th} computer should also include the update traffic (generated by all the computers) for the i^{th} computer. For those files whose directory information has already been appended to the local directory at the i^{th} computer, there is no need for the i^{th} computer to consult the master directory about these files. Thus the query response time reduces to $t_{i,d}=t_1(i)$.

For multiple master directory systems, since the directory queries are shared by the multiple master directories, the waiting time for processing the directory queries at each master directory is much lower than the single master directory system. Therefore, the response time for the multiple master directory is lower than that of the single master directory system. The query arrival rate at the k^{th} master directory d_k is $\lambda_1=\lambda_{d_k}=\sum_{\{i\}\epsilon d_k}q_i+\sum_{i=1}^{n}p_iq_i$ and the query reply rate at the output queue for the i^{th} computer is $\lambda_2=q_i$. Thus the response time for the i^{th} computer to the k^{th} master directory can be computed from (19) by replacing the $\sum_i q_i(1+p_i)$ in the first term of (19) by $\sum_{\{i\}\epsilon d_k}q_i+\sum_{i=1}^{n}p_iq_i$. When directory queries are generated by those computers that store the master directories, these query replies do not require transmission. Thus the response time equals $t_1(d)$.

For the local directory system whose replies are returned directly to the sender (i.e., without routing), the i^{th} computer may locate the information of the file before reaching the k^{th} local directory. Therefore on the average only half of the queries generated at the i^{th} computer will reach the k^{th} computer. Thus the query arrival rate at the input queue of the k^{th} local directory is $\lambda_1=\frac{1}{2}\sum_{i=1}^{n}q_i$, where q_i = directory query rate generated by the i^{th} computer. Assuming there is an output queue for each destination at the k^{th} computer, then the arrival rate at the output designated for the i^{th} computer is $\lambda_2 = q_i$. Substituting λ_1 and λ_2 into (18), we have

$$t_{i,k} = \frac{\sum_{i=1}^{n} q_i}{4\mu_1\left(\mu_1 - \frac{1}{2}\sum_{i=1}^{n} q_i\right)} + \frac{lq_i}{2R[R/l - q_i]} + \frac{2l}{R} \tag{20}$$

Because the local directory system uses routing, queries are passed around from one directory to another rather than sent directly to the sender. With a carefully designed routing strategy, the input traffic rate λ could be greatly reduced. Further the third term of (20) reduces to half of its value. Therefore, the query response time for the system with routing could be much smaller than that without routing.

Assuming all the files in the data base are equally likely to be stored in any local directory system, the time to locate a file by the i^{th} computer is

$$t_i = \frac{1}{2}\sum_{k=1}^{n} t_{i,k} \tag{21}$$

and the real expected directory response time for the i^{th} computer is

$$t_i^r = \frac{1}{2}\sum_{k=1}^{n} [t_{ik} + t_4(k)]$$

For the distributed directory system, each computer has a master directory. Public file information can be obtained from the file directory at the user's computer. Thus, $t_2(i) = t_3(i) = 0$. The query arrival rate at the directory input queue is $\lambda_1 = q_i\left(1 + \sum_{k=1}^{n} p_k\right)$. The expected directory query response time reduces to

$$t_{i,i} = t_1(i) = q_i\left(1 + \sum_{k=1}^{n} p_k\right) \Big/ \left\{2\mu_1\left[\mu_1 - q_i\left(1 + \sum_{k=1}^{n} p_k\right)\right]\right\} \tag{22}$$

NUMERICAL COMPUTATIONS

From the models we developed in the last section, we know that the operating cost of the directory system depends on many parameters such as network topology, transmission cost, storage cost, translation cost, directory query rate, directory update rate, and directory size. Although the models are formulated in general terms and can be applied to different parameter values, in order to draw some meaningful conclusions as well as limit the number of study cases, we shall select two types of network topologies: a star network which has the property that the distances from the center node to all the computers are equal, and a distributed network of computers that forms clusters, the distances among these clusters being unequal. In our example we assume that all the computers in the system have the same storage cost, code translation cost, and directory update rate; the size of the master file directory is directly proportional to the number of computers in the system; all communication channels have full duplex capabilities and the same communication cost. First, we study the star network as shown in Figure 1a. We computed the monthly operating cost as a function of file directory query rate and directory update rate for various directory systems. Numerical results are shown in Figures 1b and 1c. The parameters used in this example are: $C^t/C^s = 10$ month/mile, $T/C^s = 2000$ byte month/transaction, $C^s = 7.0 \times 10^{-5}$ \$/byte month, $F = 6 \times 10^5$ bytes, $F_1 = 3 \times 10^3$ bytes, $l = 50$ bytes, $\alpha = 0.1$, $\beta = 1/3$, and $q = q_i = 1000$ queries/month. The directory update rate is expressed in terms of the normalized directory update rate which is the ratio of directory update rate to directory query rate.

In the same manner, we study the distributed network which consists of ten computers that forms into three clusters as shown in Figure 2(a). Using the same parameter values as for the star network, we compute the monthly operating cost as a function of directory query rate and normalized directory update rate. The results are given in Figures 2(b) and 2(c).

So far we have assumed that each computer has the same directory query rate. We shall now relax this assumption and let the query rate at each computer be different. In order to compare with the uniform query rate case, we let the total number of directory queries generated from all the computers remain the same but vary the query rates among the computers. For example, we let computers 1, 2, 3, 4, and 5 have a rate of 1500 queries/month and others 500 queries/month. The results reveal that the operating cost of the centralized directory system with the above stated query rate is lower than that of the case that 1000 queries/month generated from each computer (Figure 2(c)). On the other hand, if we interchange the directory query rate of computers 1 to 5 with computers 6 to 10, then the operating cost of the centralized directory system with the above non-uniform query rate is higher than that of the uniform query rate case.

Next, we study the operating cost characteristics of the multiple master directories system. According to the distance among computers, we partitioned the ten computers into three groups in which each computer belongs to one of the three master directories. We evaluate the monthly operating costs as a function of directory update rate from Equation (5) and compare them with those of the extended centralized directory system, the local file directory system, and the distributed directory system as shown in Figure 3.

Figure 4 shows the operating cost trade-offs for file directory systems as a function of the ratio of communication cost to storage cost. The results are obtained from the intersection of the operating cost curves un-

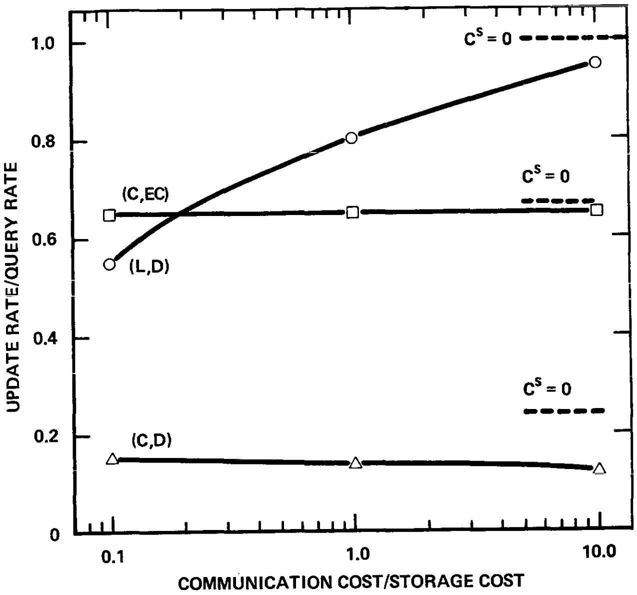
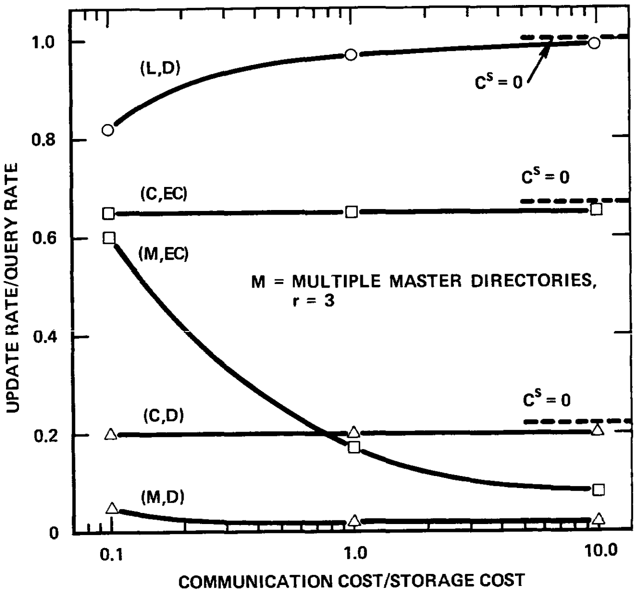


Figure 4—Cost trade-offs for various directory systems. (x,y) = cost trade-offs for directory systems x and y—(a) Star network

der various communication cost to storage cost ratios.

Finally, we study the query response time for the three classes of directories. The time between interrupts to process the directory is assumed to be equal to the time required to transmit the query; that is, $1/\mu_1 = 1/\mu_2 = 1/R$, where R is the transmission rate of the communication channel. Numerical results of the delay occurring at the directory input queues, directory query response time, and the traffic intensities with $R = 960$ char/sec are portrayed in Figure 5.



(b) Distributed network

DISCUSSION OF RESULTS

From the numerical examples studied in the last section we notice that the operating cost of the file directory depends greatly on the directory query rate and the directory update rate. Because of the large amount of data communication and translation associated with the directory updates in the distributed directory system, the rate of increase in operating cost with respect to directory update rate for the distributed directory system is higher than that of the centralized directory system. In the local directory system, we only need to update the local directory that generates the update and no transmission is required. Therefore, the operating cost is independent of the directory update rate. For a given network topology the operating costs of the local directory systems are higher than the centralized directory systems.

Assuming that the transmission cost is higher than the storage cost, our study reveals that when the directory update rate is low (e.g., less than 10% of the query rate), the distributed directory system yields lower operating costs than the centralized directory system. As directory update rate increases, the oper-

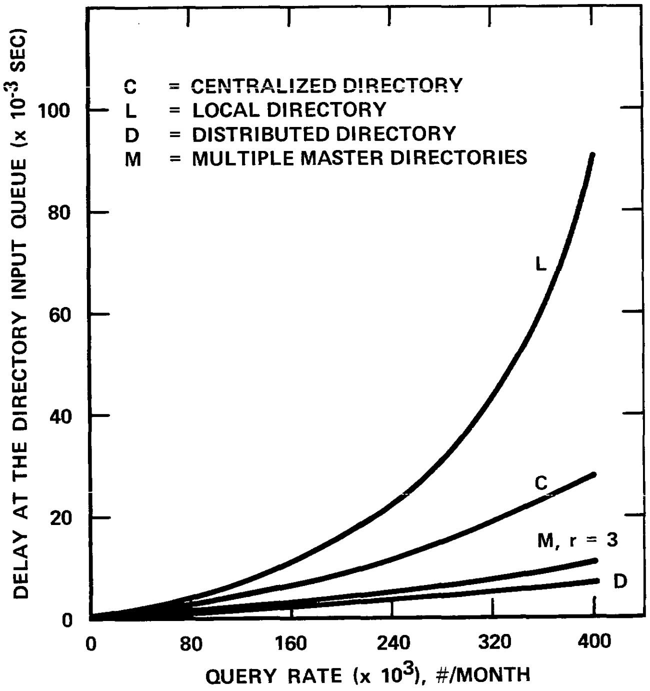
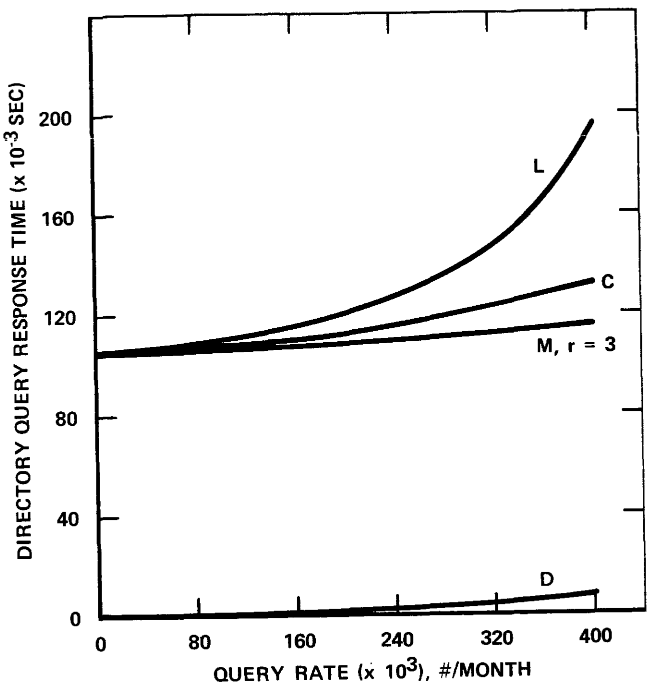


Figure 5—Expected queuing delay time for various directory systems—(a) Expected delay at the directory input queue vs. directory query rate. Normalized directory update rate=0.5

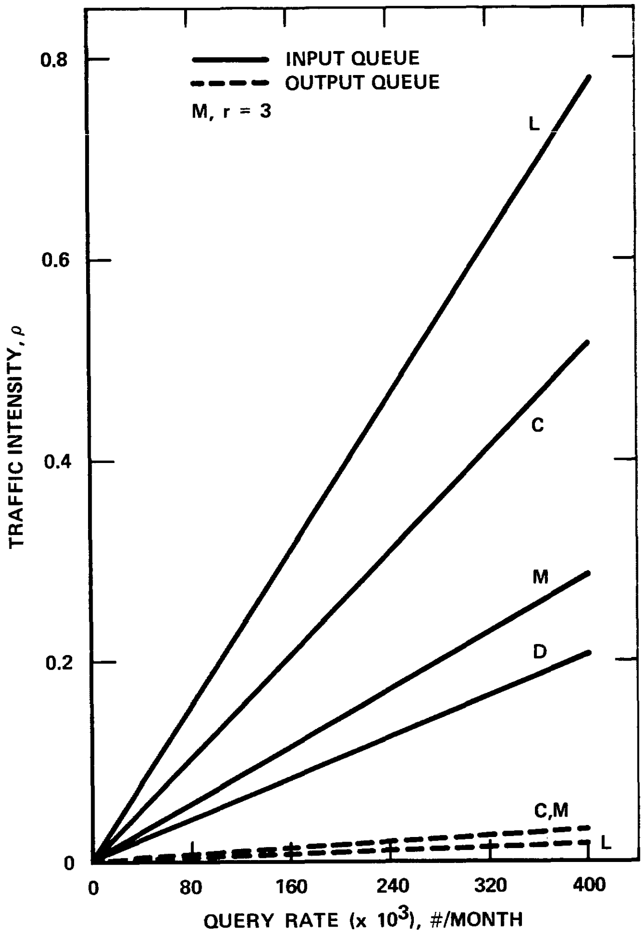


(b) Expected directory query response time vs. directory query rate. Normalized directory update rate=0.5

ating cost of the centralized directory system yields lower operating costs than the distributed directory system.

Comparing the two types of centralized directory systems, the extended centralized directory yields lower operating costs than the centralized directory at low directory updating (less than 50% of the query rate), and the performance reverses at high directory update rates. This is because of the excessive data communications required in the extended centralized directory system to update all the extended local directories. The exact cross-over point of the operating cost curves for these two types of directory systems depends on network topology and such parameters as storage cost, transmission costs, translation costs, etc. As the directory update rate increases, the performance characteristics of the extended centralized directory system become similar to that of the distributed directory system.

We also studied the influence of the distribution of the directory query traffic on the operating cost. In order to provide a common base for comparison, we kept the total number of queries generated by the computers to be a constant, and varied the query traffic among the computers. We found that the traffic distribution does not have an effect on the directory operating cost when all the computers are equal distances from each other, and does have an effect on the operating cost when the distances among the computers are different.



(c) Traffic intensity vs. directory query rate. Normalized directory update rate=0.5

When a network of computers forms in clusters (Figure 2(a)) and when their directory update rates are low, our studied example reveals that installing a master directory at each cluster requires less communication cost and therefore yields better performance than the extended centralized directory system as shown in Figure 3.

Figure 5(a) displays the queuing delay occurring at the input of the directory. The queuing delay increases as the query rate increases. Except in the local directory case, the queuing delay increases as the directory update rate increases. This is because the update messages are considered as input traffic to the directory. Since the input traffic to the multiple master directory system is shared among the master directories, it yields lower queuing delay than the centralized directory system. In the distributed directory system, the input traffic consists only of queries generated from its initiator and the directory updates generated from the rest of the computers in the system. Therefore such systems have the lowest delay at the directory input queue. Figure 5(b) displays the query response time for different types of directory systems. For the range of query rates that we have studied, the time spent in transmission to and from the directories constitutes a large portion of the delay. Since the distributed directory system does not require such transmission, it yields the lowest query response time. In the extended centralized directory system, the directory output traffic also has to include directory update traffic; therefore, the query response time of the extended centralized directory system has similar characteristics but slightly higher than those of the distributed directory system. Figure 5(c) displays the traffic intensity of the directory input queue and the directory output queue as a function of query rate. The traffic intensity provides an indication of the level of traffic at input queue and output queue(s) at the directory.

CONCLUSION

Based on the computer network topology, communication cost, storage cost, code translation cost, directory query rate, and directory update rate, we studied the cost-performance tradeoffs of the three classes of directory systems. Assuming that the transmission cost is much higher than the storage cost, our study reveals that for low directory update rates (less than 10% of the query rate), the distributed file directory yields a lower operating cost than the centralized directory system. When the update rate is greater than about 15% of the query rate, the centralized directory yields a lower operating cost than the distributed file directory system and also the local directory system. The extended centralized file directory system yields a lower operating cost than the centralized directory systems at low directory update rate (for example, less than 50% of the query rate). For a system which has a very low directory update, for example, less than 5-10% of query rate, the extended centralized directory system should be used. When a network topology forms in clusters and when directory updating is greater than 5-10% of the query rate, multiple directories systems with a master directory at each cluster yield a lower operating cost and directory response time than the extended centralized file directory system.

In the local directory system, because of the large amount of communication costs associated with searching a file when it is not stored in the user's local file directory, the operating cost is much higher than that of the centralized directory system. However, when the directory update rate is very high (greater than 50% of the query rate) and when communication cost is lower than the storage cost, the local directory system yields a lower operating cost than the distributed directory system. Since the local directory system requires a large amount of communications as well as search time for locating a file, efficient routing strategy could greatly reduce the operating cost and query response time in such systems.

We have also studied the cost tradeoffs when the storage cost is equal to or greater than the communication cost. Our study reveals that the cost curves are similar to the cases where the communication cost is higher than the storage cost except the intersection points of these cost curves differ as shown in Figure 4.

Since the distributed directory system and the extended centralized directory system (assuming that the requested file resides in the appended local directory at the user's computer) do not require communication for querying the directory, they yield lower file query response time than those of the centralized directory system and the localized directory system. When using the multiple master directories, the queries generated by the users are shared by the master directories in the system. Therefore, the directory query response time for the multiple directory system is lower than that of the single master directory system.

For a given network topology and operating environment, we can use the models developed in this paper to study the operating cost tradeoffs and directory response time for various directory systems. Such investigation provides us with a guide in designing directory systems for distributed data bases.

ACKNOWLEDGMENTS

The author wishes to thank E. E. Nahouraii of IBM for his stimulating discussion in the formulation of the model and David Lee and Shaila V. Kinariwala of UCLA for their assistance in performing the computation.

REFERENCES

1. Chu, W. W., "Optimal File Allocation in a Multiple Computer System," *IEEE Transactions on Computers*, October 1969, pp. 885-889.
2. Whitney, V. K. M., *A Study of Optimal File Assignment and Communication Network Configuration*, Ph.D. dissertation, University of Michigan, 1970.
3. Casey, R. G., "Allocation of Copies of a File in an Information Network," *SJCC 1972*, AFIPS Press, Vol. 40, 1972.
4. Chu, W. W. and G. Ohlmacher, "Avoiding Deadlock in Distributed Data Bases," *Proceedings of the ACM National Symposium*, Vol. 1, March 1974, pp. 156-160.
5. Kleinrock, L., "Queuing Systems," Vol. 1, *Theory*, Wiley-Interscience, 1975 pp. 188-190.

On measurement facilities in packet radio systems*

by FOUAD A. TOBAGI, STANLEY E. LIEBERSON and
LEONARD KLEINROCK
University of California,
Los Angeles, California

ABSTRACT

The growth of computer networks has proven both the need for and the success of resource sharing technology. A new resource sharing technique, utilizing broadcast channels, has been under development as a Packet Radio system and will shortly undergo testing. In this paper, we consider that Packet Radio system, and examine the measurement tasks necessary to support such important measurement goals as the validation of mathematical models, the evaluation of system protocols and the detection of design flaws. We describe the data necessary to measure the many aspects of network behavior, the tools needed to gather this data and the means of collecting it at a central location; all in a fashion consistent with the system protocols and hardware constraints, and with minimal impact on the system operation itself.

INTRODUCTION

This paper is primarily concerned with the unique measurement aspects of Packet Radio Systems as regards network evaluation, and considers the design of a set of measurement facilities, the development of data gathering techniques within the framework of the system design and the use of these measurements to evaluate the system performance and its operational algorithms.

The need for sharing of computer resources by organizing these resources into computer networks has been long recognized[1] and the feasibility of constructing such networks has been demonstrated by many successfully operating network systems. Perhaps the most prominent example is the ARPANET,[2] which utilizes the technique of packet-switching, appropriate for bursty computer network traffic, thus achieving better sharing of the communication resources.

The ARPANET emerged in 1969 as the first major packet-switching network experiment; since the essence of an experiment is measurement, and in line with Hamming's observation that "it is difficult to have a science without measurement", considerable care was taken from the beginning in the design and development effort to include the tools necessary and appropriate to satisfy the many measurement goals. As a result of well designed experiments on the ARPANET using these tools, valuable insight has been gained regarding the network usage and behavior.[3]

The Packet Radio System is another yet different example of a computer resource sharing network.[4] It is being developed by the Advanced Research Projects Agency in order to demonstrate the applicability of the packet radio concept in organizing computer resources into a computer communications network. It is this packet radio network which is of concern to us in this paper. The network is currently in its design phase,* and, as was the case with the ARPANET, care is being taken to include the ability to measure network behavior. UCLA is in charge of this measurement effort.

This concern for measurement is due to several factors. Firstly, these measurements provide a means to evaluate the performance of the operational protocols employed and the identification of their key parameters. Moreover, this realistic observation of the system behavior will assist in the validation and improvement of existing analytical models devised to study some of these operational schemes, such as the access modes and routing strategies.[5,6] Secondly, these measurements will allow for the detection of system inefficiencies and the identification of design flaws such as the inadvertent creation of a deadlock condition.[7] Thirdly, measurement facilities and data, when used to improve network design, are a valuable feedback process in which design deficiencies are detected and subsequently corrected. Wire networks differ from radio networks mainly in the omni-directional broadcast nature of the communication and consequently the protocols employed; therefore, it calls for new approaches in the design and implementation of the measurement facilities and their use.

* This work was supported by the Advanced Research Projects Agency of the Department of Defense (DAHC-15-73-C-0368).

* A preliminary demonstration of the system is under way. A prototype network is being set up in the Palo Alto, California, area.

In the following section, we present an overview of the packet radio system concepts and a brief description of the currently specified operational procedures.

In a later section, we describe the network measurement facilities which consist of the measurement tools and the techniques for data collection. In the last section, we identify and discuss in some detail the desirable measurement functions to satisfy the need for validation and performance evaluation outlined above.

THE PACKET RADIO SYSTEM

Several papers have already appeared in the literature which describe the packet radio concept and discuss many of the issues involved in the system design.[4-6,8-10] In this section, we briefly describe these system components and operational procedures necessary to understand the measurement considerations presented below.

There are three basic functional components of a packet radio system:

(i) packet radio terminals—these are the sources and destinations of traffic on the packet radio network.

(ii) packet radio stations—these function as S/F switches for local traffic and as interfaces between the broadcast system and other computers or networks. Also, they perform directory, monitoring and control functions for the overall system, and they play a central role in that all traffic passes through the station, i.e., we have a centralized network.

(iii) packet radio repeaters—their function is to extend the effective range of terminals and stations by acting as Store-and-Forward relays.

The repeater, which has been developed by Collins Radio and is called a packet radio unit (PRU), consists of a radio transceiver and a microprocessor. The function of the PRU is to receive and transmit packets according to dynamic routing and control algorithms specified by the station. For simplicity and uniformity of design, the PRU is used as the front-end of terminal devices and of stations, interfacing them with the radio net. In Figure 1 we show an oversimplified picture of the PRU identifying its various sections: the radio transceiver, the store-and-forward software, the control process, and the measurement process.

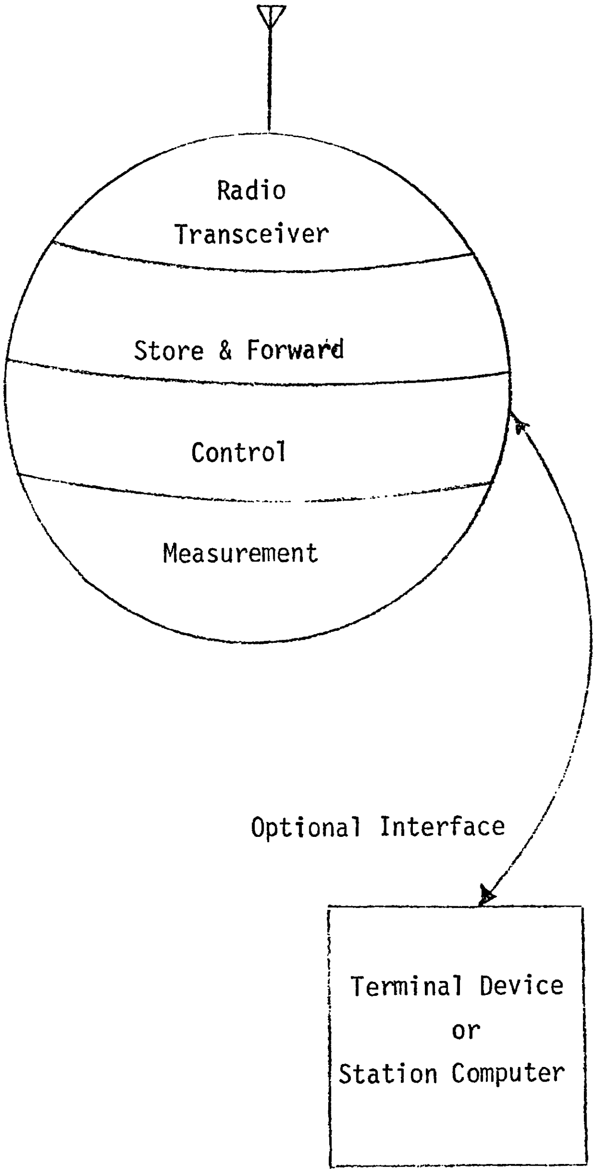


Figure 1—The packet radio unit

In this initial system, the terminals, stations and repeaters are linked together by a single broadcast channel using omni-directional antennas. The repeaters do not determine routes. All the routing computations are performed by the station. A hierarchical routing algorithm is used which makes the routing in the broadcast network resemble routing in a point-to-point network by forming a hierarchical tree structure. This structure is constructed by having the station assign to each repeater a label which defines its position in the tree. A packet is routed along the path determined by the tree, requiring the packet header to contain a string of appropriate repeater ID's, or labels. Thus, neighboring repeaters hearing the broadcasted packet but not on the determined path will reject the packet rather than relay it. However, this algorithm is flexible in that it allows the repeater to seek an alternate route for a packet when a path seems to be blocked. Moreover, the station with its monitoring procedures can dynamically restructure the tree by relabeling any of the repeaters in response to component failures or traffic congestion.

In order to achieve reliable packet transport, acknowledgment procedures are required. There are two types of acknowledgments; the end-to-end acknowledgments (FTE) between end devices, and hop-by-hop acknowledgments (HBH) between repeaters.[6] Except for the last hop on a packet's route, HBH acknowledgments are passive in that the relaying of a packet over a hop constitutes an acknowledgment of the transmission over the previous hop; this "echo acknowledgment" is due to the omni-directional broadcast property. At the last hop, an *active* HBH acknowledgment must be generated.

MEASUREMENT FACILITIES

Several factors exist in the packet radio system which do not allow for a simple transfer of ARPANET-like measurement facilities to a packet radio network. Although the latter utilizes the same technique of packet-switching, the packet radio concept is unique in the constraints it places on all system operations and the measurement effort in particular.

The radio broadcast nature of transmissions is such that the transmission of measurement data not only introduces overhead over its own path, but causes transmission interference at neighboring repeaters within hearing distances and creates additional overhead on those PRU's activities. Moreover, the desire to keep the components small and portable, as well as the limited speed of the IMP's CPU within the PRUs, place significant constraints on the measurement facilities and their usage. The available storage is extremely limited and the overhead placed on the PRU's CPU is of utmost importance in evaluating the feasibility of a measurement tool and of the collection of data in support of a measurement function. As the operational protocols of the net are different from wire nets, the measurement functions devised to support the evaluation of their performance are unique. Thus, the measurement effort consisted of identifying the measurement functions (as described in the following section) and devising the measurement facilities required to support those functions under the constraints that the system imposes. The development of the tools was an iterative design process seeking a balance between supporting the measurement functions and satisfying the system constraints, as well as making sure that the network communication protocols allow the implementation and proper functioning of those tools.

In this section, we describe the various types of statistics desired in the Packet Radio Net,* the traffic sources required in measurement experiments and the techniques available for measurement data collection. We shall postpone until the next section the detailed list of the quantities that will be measured by each of the types of statistics (tools).

Cumulative statistics (Cumstats)

As its name suggests these consist of data regarding a variety of events, accumulated over a given period of time, and provided in the form of sums, frequencies and histograms. We shall distinguish between those data collected at the PRUs (PRU based Cumstats) and those collected at the end devices (the end-to-end Cumstats). The PRU based Cumstats provide information about the *local* environment and behavior such as traffic load, channel access, routing performance, and repeater activity. Conversely, end-to-end statistics collected at network sources and sinks, that is stations and terminal devices, will reflect more global network behavior such as user delays and network throughput.

Trace statistics

The trace capability allows one to literally follow a packet through the network, and to trace the route which it takes and the delays which it encounters at each hop. In the ARPANET, selected IMPs gather data on packets to be traced (which may include any packet) and send this data to the collection point as a new packet. In the packet radio network, however, the collection of trace data at the repeaters is prohibited by the limited size of storage in the PRU. To overcome this problem, we have introduced a new type of packet called the Pickup Packet.* These packets are generated with an empty text field by traffic generators at end devices. As these packets flow normally in the network according to the transport protocols, selected repeaters will gather the trace statistics and will store them within the text field of the pickup packets themselves.

Snapshot statistics

Snapshots give an instantaneous peek at a PRU, showing its state at that moment with regard to buffer assignment and queue lengths. (In the ARPANET, which is a decentralized network in which each node contains routing algorithms and data, snapshots also include routing related information; in the Packet Radio Network, such information is available at the station). Changes to appropriate station tables will be time stamped and collected as the station's snapshot function.

Artificial traffic generators

Traffic sources

The creation of streams of packets between given points in the net, with given durations, intervals,

* These types of statistics, as well as traffic generators, which have been widely used in ARPANET measurement experiments, will differ significantly from those of the Packet Radio Network in regards to the specific quantities gathered and the means of collecting them at a central location.

* The notion of the pickup packet was first suggested by H. Opderbeck.

packet lengths, and packet types (Information and Pickup Packets) is clearly a requirement of any experimental system. While it might be desirable to provide each PRU with the capability of creating such traffic, this additional burden on the PRU software can be avoided if there exists a reasonable number of terminals with processors attached which, along with the station, will be programmed to provide the traffic-source functions indicated above.

Specifically, traffic-source features which the terminals (and Station) should provide are: (1) Information Packets—the user specifies the packet length, frequency, destination and duration of one or more streams of Information Packets. (The text content may be arbitrary.) (2) Pickup Packets—the user specifies the packet length, frequency, destination and duration of one or more streams of Pickup Packets.

In the initial system, there will be a limited number of system elements, making it desirable to simulate in a terminal a multi-terminal environment. That is, the traffic generated at a single terminal will emulate the traffic that would be generated by several separate sources. A great deal of complexity is introduced in the design of these devices because of the hardware and software capabilities required to support this function. Feasibility and techniques of achieving this is under investigation.

Station measurement process

Since the station is the central node and provides central control for the operation of the entire network it therefore plays a central role in the execution of measurement functions. It is through the station that the initiation and termination of measurement experiments is controlled. In particular, the station enables and disables the Cumstat and Pickup packets functions at the PRU's, and assigns to the various elements the intervals for Cumstat collections, and to the artificial traffic generator, their corresponding parameters. Moreover, it is to the station that all measurement data is ultimately destined; upon arrival at the station, the data is time-stamped and stored in a single measurement file for off-line reduction and analysis. In addition, all changes to the station's internal tables (routing, connectivity, PRU operational parameters, etc.) will be reflected by an entry into the measurement file, thus allowing the correlation of measurement results to the actual network configuration. A (measurement) process at the station will perform all of the above functions.

Measurement data collection

As mentioned earlier, pickup packets are generated at stations and terminals. Those packets generated at a terminal are destined to (and collected at) the station; those generated by the station will be returned by their destinations to the station as regular packets for collection into the measurement file.

Let us now discuss the techniques for centrally collecting cumulative statistics. The data, generated at the PRU's or terminal devices, must be transmitted to the station using the PR Net itself. One way of achieving this is to form at the PRU, at the end of each Cumstat interval, a measurement packet called the Cumstat packet, which is time-stamped and transmitted to the station. The second method consists of having the station send at regular intervals to appropriate PRU's an executable control packet* called an Examine packet which collects time stamped Cumstat data and which returns back to the station.

For purposes of analysis, it is desirable for the Cumstat data received at the station to correspond to equal length time intervals at the generating device. This can be achieved in the automatic method if reliable ETE transmission exists, i.e., if ETE acknowledgment capabilities are provided in all terminal devices and PRU's, preventing the loss of a Cumstat packet from a device on its way to the station. In the absence of the ETE capability in the PRU's, one may decrease the Cumstat intervals (thus increasing the frequency of transmitting Cumstat data), thereby decreasing the gaps between correctly received Cumstat packets. With the Examine method, variable length Cumstat intervals will occur since Examine packets, sent at regular intervals from the station, are subject to (i) the network random delays en route to the destination PRU, and (ii) the possibility of loss in either direction.

The choice of a collection method will have to take into consideration the overhead that it imposes on the PRUs and on the network.

MEASUREMENT FUNCTIONS

We have described in the previous section the measurement facilities that are desirable in a PRNET to support the measurement functions. In this section we shall identify and discuss these functions in some detail, determine the required data items and describe the role of these measurement facilities in supplying the data. These include: channel access, operational protocols, repeater performance, traffic characteristics, and the network's global performance.

Channel access

One of the main features that distinguish the Packet Radio Network from point-to-point networks is that

* An executable control packet is a packet that originates at the station and is destined to a PRU. It contains code to be executed by the destination PRU. In particular, the Examine packet contains the necessary code to load the contents of specified memory locations into the text of the packet for shipment back to the station.

devices transmit packets over a broadcast channel by using a random access scheme. These random access schemes are characterized by the sharing of a single channel in a multi-access fashion, thus allowing for packet interference to occur. Considerable progress has been made in analyzing these access modes, which include pure and slotted ALOHA and the more recently developed techniques of Carrier Sense Multiple Access (CSMA).[5,11-13] In the initial experimental system pure ALOHA, 1-persistent CSMA and non-persistent CSMA will be available. Our measurement aims are to validate the analytical models of the three access modes and to evaluate their performance in realistic environments.

In evaluating terminal access in a single hop system (a model commonly used in analysis), we consider an environment consisting of a single station and a population of terminals within range and in line-of-sight of the station. In order to determine the relationships between the network throughput (rate of successfully received packets at the station) and channel traffic (rate of packet transmissions over the channel), as well as the relationships between the network throughput and packet delays, the following quantities will be measured:

(a) the number of transmissions a packet incurs before success

(b) the one-hop packet delay: time elapsed since the packet is ready for transmission until it is acknowledged, i.e., until its acknowledgment packet is received from the station

(c) network throughput: average number of packets received at the station per unit time

Items (a) and (b) are obtained in the form of histograms by the Cumstat tools at the PRU and the end device respectively. Item (b) may also be obtained individually for each Pickup packet by having the originating device store in it its time of generation and its transmission times, and in the succeeding Pickup packet, store the time its acknowledgment arrived. Item (c) is obtained at the station from end-to-end cumulative statistics.

The task of measuring performance of terminal access techniques in multi-repeater environments differs from the previous one in that repeater-to-repeater traffic is present contending on the same channel. The environment consists of a number of repeaters and stations and a population of terminals, not necessarily all within range and in line-of-sight. The same quantities as listed above, measured over the terminal-to-repeater hop, will be collected using the same tools.

Operational protocols

Acknowledgment protocols

Echo acknowledgment suffers from packet interference. The delay until the echo acknowledgment is received at the transmitter is random. Thus, the packet may incur some additional transmissions beyond the first successful one creating additional overhead on the channel and in the PRUs. This number of additional transmissions is a measure of the inefficiency of echo acknowledgments; so too will be the number of packets discarded at the transmitter because of lack of reception of the echo acknowledgment. That is, the transmitter reached the maximum number of retransmissions of a packet before the echo acknowledgment was received; although the packet may have been successful, the transmitter declares itself unsuccessful in establishing communication!

Thus, we shall measure the efficiency of the Echo Acknowledgment protocol by measuring the number of additional transmissions beyond success incurred by a packet. To compute this number, a PRU must have two pieces of information; it must know how many times the packet has been transmitted, and it must also know which of those retransmissions was the one that reached the next repeater successfully. This information will be contained in two fields in each packet header, which we refer to here as fields A and B. Field B is used by the PRU to store the current transmission number of the packet. When the packet is successfully heard by the intended receiver, the contents of field B are saved by being stored into field A; when the Echo acknowledgment is successfully heard by the sending PRU, field A of the echo acknowledgment is compared with the current number of transmissions of the packet, i.e., the contents of field B in the sender's copy of the packet. If these two numbers differ, then the magnitude of that difference represents the number of times that the packet was retransmitted after it had already been successfully received at the next hop. This data is collected as part of the cumulative statistics of the sending PRU.

Routing protocols

Earlier we introduced the hierarchical routing scheme in use, which is based on a tree structure with the station as its root. The initial tree structure is created dynamically by the Initialization Procedure in which the station uses PRU connectivity information to create a tree that minimizes the number of hops between each repeater and the station. Thus the routing strategy initially performs shortest path (minimum hop) routing from repeaters to station and from station to repeaters. However, when the first choice shortest path cannot be used, the packet departs from this path and uses a shortest path from its new location. This will occur when a repeater has transmitted a packet over a hop the maximum number of times allowed without receiving an HBH acknowledgment; the repeater then alters the packet's header (to what is called the "ALL" label) so that any repeater within hearing distance and able to relay the packet in its intended direction will do so. This packet is then said to be alternately routed. It is retransmitted with its "ALL" header until either an HBH acknowledgment is received or the maximum number of retransmissions is once again reached, at which time the packet is discarded.

The analysis of a routing algorithm, particularly in a broadcast, and thus mobile, network, is a complex task, in that routing is topology- and load-dependent, and involves, with varying degrees of subtlety, all of the system's protocols. Thus, routing considerations are really a synthesis of most elements of the system design, and as such, the measurement of the algorithm involves at times the study of the interaction of the many system protocols.

Given the patterns of input load on the network, the *distribution of traffic flow* in the net is an indication of the behavior and efficiency of the routing and initialization algorithms. One may detect the concentration of traffic on specific routes creating congestion while alternate routes are not assigned; thus smaller delay routes may have been ignored in favor of the shorter routes provided by the initialization procedure.

To obtain the distribution of traffic flow, the following quantities are to be measured.

(a) the total number of packets received and transmitted at each repeater (obtained in the PRU Cumstats)

(b) the fraction of time the transceiver is busy (obtained by snapshot statistics, or in the PRU Cumstat by regular sampling of the transceiver's state)

Also, the point-to-point nature of this routing algorithm, restricting a packet at a given hop to a single repeater as its immediate destination, does not take advantage of the broadcast nature of the channel, in which several neighbors may actually hear the transmission and be capable of relaying the packet. Thus the following quantity is relevant:

(c) the number of packets correctly received and discarded because they are destined to other components in the net (obtained in the PRU Cumstat).

Moreover, to measure the potential of each neighboring repeater (say, repeater "n") as an immediate destination, it is essential to know the probability of success P(n) repeater n has to correctly receive a broadcast packet. This we do by maintaining in each PRU a table counting the number of successfully received packets from each immediate neighbor. The ratio of the number of packets correctly received from a given neighbor, to the number of packets transmitted by that neighbor, is a measure of P(n).

Another important feature of a routing algorithm is its adaptability to network changes: input traffic load, connectivity and component failure and repair. In evaluating the dynamics of such an algorithm, three factors must be examined: the time required to detect the network change, the time required to respond, and the quality of the response. The data items at each PRU necessary for these studies, which include some of those mentioned earlier, are:

(a) tables counting the number of packets correctly received from immediate neighbors

(b) number of packets alternately routed

(c) number of packets discarded, suggesting route congestion or component failure

(d) percent of time repeater is busy transmitting and receiving

These are obtained as cumulative statistics in the PRU.

In addition, the Pickup packet is a valuable tool in routing studies in that it contains the actual and complete route taken by the packet (pinpointing alternate routing), as well as time stamps to compute the queueing and transmission delays incurred at each repeater.

Repeater's performance

The evaluation of the performance of a repeater is most important in the analysis of network behavior; it allows us to break down key network measures (such as packet delay and throughput) into their elementary components and to examine the effects on these measures of the repeater activity and design (including buffer management, queueing discipline, and packet processing priorities).

The quantities relevant to packet delays are:

(a) The processing time of a packet flowing through a repeater; this is counted in Pickup Packets as the time lapse between the packet's arrival and the time it is placed on the transmission queue. This processing includes various checks such as checksum, packet type, routing labels, etc.

(b) the packet queueing delay at a repeater; this is also counted in Pickup Packets as the time elapsed from when the packet is placed on the transmission queue until it is considered for transmission (i.e., until it is at the head of the line, in a first-come-first-served discipline).

(c) the packet's service time; this is also counted in Pickup packets as the time elapsed from when the packet is at the head of the queue until its echo-acknowledgement is correctly received. Note that the actual service time (time until the packet is correctly received at the next repeater) is smaller than the one measured here due to the echo acknowledgment protocol used in this system. Note also that the service times of consecutive packets are correlated.

The quantities related to a repeater's communications activity are:

(d) percent of time the PRU transceiver is busy transmitting and receiving; this can be obtained in the PRU

Cumstat by regular sampling of the transceiver's state.
(e) the total number of transmitted packets at each repeater relative to the number of successfully transmitted packets. The latter number is obtained for each neighboring repeater by examining its table count which gives the number of packets correctly received from immediate neighbors.
(f) the percent of traffic received with checksum error (obtained in the PRU Cumstats).
(g) the percent of traffic received correctly but not intended for this repeater (obtained in the PRU Cumstat).

The quantities relevant to buffer management and occupancy are:

(h) the percent of time packet buffers are in a given state (free, queued for packet transmission, reserved for packet receive). This can be obtained in the PRU Cumstat by a regular sampling of the buffer states.
(i) the frequency of buffer overflow as a function of the load, and this is obtained also in the Cumstats by counting the number of packets discarded due to lack of buffer space.

Traffic characteristics

In determining the traffic characteristics, one should distinguish between *external traffic* (the input traffic generated by network users and traffic sources) and *internal traffic* (traffic relayed and generated at repeaters). The measurement functions determining the external traffic characteristics are not necessary when the entire traffic is artificially generated. They include:

(a) the geographical distribution of the input load (obtained in the end device Cumstats)
(b) characteristics of the terminal input processes (obtained in the form of histograms of packet intergeneration time from the end device Cumstats)
(c) the amount of traffic generated at repeaters for special purposes such as: control, measurement, etc. (i.e., overhead traffic) (obtained in the PRU Cumstats)

The characterization of internal traffic is crucial in the creation and validation of assumptions made in repeater models aimed at an analytic prediction of the performance of multi-repeater packet radio networks. To characterize this internal traffic, we may measure the following quantities at each repeater:

(a) interarrival time (defined as the time between the arrivals of two successive packets that have been correctly received and are destined to that repeater).
(b) interdeparture time (defined as the time elapsed between the acknowledgment of two consecutively transmitted packets).

Histograms of these quantities can be created from the information contained in the Pickup Packets.

Network's global performance

The ultimate goal of all system considerations is to create a network of high capacity providing minimal user (end-to-end) delay. We examine the success in achieving this goal by measuring the end-to-end delay and the network throughput (counted as the number of packets received at their respective destinations), under various patterns of input load, as well as the frequency of lost and duplicated packets.

It is important to note that these quantities are fundamentally affected by all the operational protocols. They allow us to obtain the main performance curves of throughput and delay.

The role of measurements in flow control

The station has the responsibility for centralized control over the entire network. To carry out this responsibility, the station requires various indications of network activity and performance. Some of this information will be acquired from incoming traffic, but much of this information must be specifically obtained by having monitoring procedures collect, from the various devices, a subset of the measurement items that have been seen presented throughout the paper.

CONCLUSION

In this paper, we have presented some of the results of our activities in the measurement aspect of the ARPA Packet Radio Project. We described the Packet Radio Network measurement facilities, consisting of the measurement tools and the techniques for data collection. We also identified and discussed the measurement functions required to gain insight into the behavior of this broadcast network. In so doing, we determined the data items required to support these functions and the means for their collection. This information is summarized in Table I.

In the design of these measurement facilities, a constant concern is to keep the overhead they create at the components and on the broadcast channel at a low level. An important activity will be to evaluate the cost of each element of the facilities in the prototype network, and to assess their impact on the network operation so as to design and conduct experiments in a manner that will minimize the bias introduced.

ACKNOWLEDGMENTS

We would like to acknowledge the Packet Radio Working Group and particularly Ralph Jones, David Retz, Ron Kunzelman and Don Nielson from Stanford Research Institute and Dick Sunlin from Collins Radio Corporation for the fruitful discussions that have taken place with them.

TABLE I—Summary of Measurement Items

Pickup Packets (at each PRU, the following data items are collected in the Pickup packet):

- time of arrival of the packet at the PRU
- time the Pickup packet was first placed on the transmit queue
- time of each transmission
- time HBH ack arrived (stored in next Pickup packet)
- the current PRU ID

PRU based Cumulative Statistics

- # of packets received in error
- # of packets received but not intended for this PRU
- histogram of # of transmissions per successful packet
- # of unsuccessful packets (dropped because of lack of ack)
- # of packets discarded because of lack of buffer space
- # of alternately routed ("ALL") packets received
- table counting number of correctly received packets from immediate neighbors
- # of transmissions beyond success
- # of packets incurring transmissions beyond success
- table sampling frequency of buffer states (and transceiver states)

End-Device Cumulative Statistics

- histogram of round-trip times
- # of packets transmitted
- # of duplicate packets detected
- # of packets discarded by the sender because of lack of ETE ack
- histogram of # of transmissions per successful (ETE) packet
- histogram of packet intergeneration time

Note: certain Cumstat items will distinguish between inbound (to the station) and outbound (from the station) traffic.

REFERENCES

1. Roberts, L. G. and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," Spring Joint Computer Conference, *AFIPS Conference Proceedings*, 1970, Vol. 36, pp. 543-549.
2. Kleinrock, L., *Queueing Systems, Vol. II, Computer Applications*, Wiley Interscience, 1976.
3. Kleinrock, L. and W. Naylor, "On Measured Behavior of the ARPA Network," in Nat. Comput. Conf., *AFIPS Conf. Proc.*, Vol. 43. Montvale, N.J., AFIPS Press, 1975, pp. 767-780.
4. Kahn, R. E., "The Organization of Computer Resources into a Packet Radio Network," in *Nat. Computer Conference, AFIPS Conference Proceedings*, Vol. 44. Montvale, N.J., AFIPS Press, 1975, pp. 177-186.
5. Kleinrock, L. and F. Tobagi, "Random Access Techniques for Data Transmission over Packet-Switched Radio Channels," in *Nat. Comput. Conf., AFIPS Conf. Proc.*, Vol. 44. Montvale, N.J., AFIPS Press, 1975, pp. 187-201.
6. Frank, H., I. Gitman and R. Van Slyke, "Packet Radio System—Network Considerations," *in Nat. Comput. Conf., AFIPS Conf. Proc.*, Vol. 44. Montvale, N.J., AFIPS Press, 1975, pp. 217-231.
7. Kleinrock, L. and H. Opderbeck, "Throughput in the ARPANET—Protocols and Measurement," *Fourth Data Communications Symposium*, IEEE Catalog Number 75 CH 1001-7 DATA, Quebec, October 1975.
8. Fralick, S. and J. Garrett, "Technological Considerations for Packet Radio Networks," in *Nat. Comput. Conf., AFIPS Conf. Proc.*, Vol. 44. Montvale, N.J., AFIPS Press, 1975, pp. 233-243.
9. Burchfiel, J., R. Tomlinson and M. Beeler, "Functions and Structure of a Packet Radio Station," in *Nat. Comput. Conf., AFIPS Conf. Proc.*, Vol. 44. Montvale, N.J., AFIPS Press, 1975, pp. 245-251.
10. Fralick, S., D. Brandin, F. Kuo and C. Harrison, "Digital Terminals for Packet Broadcasting," in *Nat. Comput. Conf., AFIPS Conf. Proc.*, Vol. 44. Montvale, N.J., AFIPS Press, 1975, pp. 253-261.
11. Tobagi, F., "Random Access Techniques for Data Transmission over Packet Switched Radio Networks," Ph.D. dissertation, Comput. Sci. Dep., School of Eng. and Appl. Sci., Univ. of California, Los Angeles, rep. UCLA-ENG 7499, Dec. 1974.
12. Kleinrock, L. and F. Tobagi, "Packet Switching in Radio Channels: Part I—Carrier Sense Multiple Access Modes and Their Throughput Delay Characteristics," *IEEE Trans. Commun.*, December 1975, pp. 1400-1416.
13. Tobagi, F. A. and L. Kleinrock, "Packet Switching in Radio Channels: Part II—The Hidden Terminal Problem in Carrier Sense Multiple Access and the Busy Tone Solution," *IEEE Trans. Comun.*, December 1975, pp. 1417-1433.

Monitoring and access control of the London node of ARPANET

by ADRIAN V. STOKES, DAVID L. BATES* and PETER T. KIRSTEIN
University College London
London, England

ABSTRACT

At University College London, we have developed a novel way of monitoring a network node using a separate processor and have applied this technique to the London node of ARPANET. A monitoring program on a PDP-9 records usage of the London-TIP via the dial-up ports and these data are sent to the Rutherford Laboratory IBM 360/195 for detailed analysis. In this paper, we describe the method we have developed and present some of the results we have so far obtained.

INTRODUCTION

The ARPA computer network[1] has been operational for over six years. During this time, there have been extensive measurements on the performance of the communications subnetwork, particularly by the Network Measurement Center at the University of California[2,3] and Bolt, Beranek and Newman.[4] There have been extensive measurements of usage of specific hosts, for example by the National Bureau of Standards. There have also been certain measurements of the network usage made for certain large applications to justify the cost of running the network. There have not, however, been any consistent measurements of network usage via one site. There are several reasons for this omission. Partly it is due to there being no mechanism by which US users of ARPANET could be forced to keep statistics of their usage, and partly it is due to there being no automatic accounting system for the use of the network.

An attempt was made in late 1974 to introduce an automatic accounting system into the subnetwork. The mechanism was that each communication computer would connect to a specific Access Control Host before it permitted a connection to be opened to any other Host. Further connections were permitted only if the correct user/password combination was given; after each session over a virtual circuit, the accounting Host was informed of the length of the connection and the number of packets transferred (unless the Access Control computer was not available, in which case the statistics were stored for later transmission). This mechanism was abandoned after a few weeks for several reasons; amongst these were the difficulty of maintaining the password file on the Access Control Host and the sluggishness of the response of the access control mechanism.

At about the same time we, at the University College London (UCL) node of ARPANET, became interested in providing access control and accounting. There were two reasons for this. Many bodies wished to analyze the extent and value of usage of the ARPANET link via the UCL node; they also wished to be able to control the access—particularly via the Public Switched Telephone Network (PSTN). Because we were a node like any other, it was not possible to put any special code into the Honeywell 316 Terminal Interface Message Processor[5] which acts as the communication processor to the UCL site. Instead, any such code had to be provided outside the subnetwork. However, since all the use of the UCL node is purely experimental, we were entitled to enforce any extra login procedures we wished onto our users. We have been looking at three different types of measurement:

(a) Characteristics of usage of Hosts via the UCL node
(b) Characteristics of the data traffic for several specific applications
(c) Overheads incurred in the different levels of protocol

This paper is concerned only with the first of the above; the others will be discussed in later papers. Only sample measurements will be presented here; fuller measurements will be discussed at the Conference and are really more appropriate to a Technical Report.[6] In order to provide meaningful statistics, we also have had to provide access controls; this subject is also considered here.

At UCL is sited a TIP; two PDP-9 computers are connected to the TIP, one of which acts as a gateway between various computer networks, in particular, between ARPA and the Rutherford Laboratory (RL) star network based on an IBM 360/195.[7] The second PDP-9 is used both as a development machine and for monitoring and access control of the ARPANET; it also provides a simple form of access to the National Library of Medicine (NLM) Medline system. It is

* D. L. Bates is now at CERN, Geneva, Switzerland.

the measurements made using this machine which are the subject of this paper.

OVERVIEW OF THE SYSTEM

The TIP has 8 slow (300/300 bps) and 1 faster (1200/75 bps) dial-up ports as well as three leased lines. The PDP-9 used for this work (known as PDP-9B) is connected to the TIP as a Host. It has 32K words of core (18 bit), a 256K disk and various other peripherals.

DESCRIPTION OF THE QUES PROGRAM

The program which performs the actual monitoring and access control is called QUES. This is one segment of a network system which runs on the PDP-9. When the system is initialized and has established communication with the TIP, QUES sets up a control connection to a specific port on the TIP; via this connection, QUES attempts to connect to other TIP ports (which are specified by entries in a disk file which may be modified easily). Each port may be in one of three states. The first is WILD, in which case there is no user connected; in this state, QUES may (and indeed does) make the connection. When a user dials in, this connection is broken and, on noting this, QUES remakes it and interrogates the user by asking for surname, TIP password and number of required Host (a typical scenario is shown in Figure 1). To save time, a user may give all three replies in response to the first question.

If the replies are satisfactory, QUES then breaks the connection and waits 20 seconds before attempting to remake it. If the replies are unsatisfactory, the user is allowed a second attempt then, if still incorrect, he is disconnected. It would have been possible for us to have better access control by making the connection for the user. This would require, however, a considerably heavier cpu load and to reduce this, we only make the connection in the specific case that we wish to record the whole dialogue for subsequent analysis (as we do in the case of MEDLINE; see below).

If the user gives correct answers and succeeds in connecting to the Host within the requisite time, QUES enters its third phase in which it attempts (at one second intervals) to reconnect to the port.

```
LONDON-TIP MONITORING SERVICE.
SURNAME >stokes
TIP PASSWORD >×××
PASSWORD UNKNOWN—REENTER >indra
HOST NUMBER >42
OK—BYE
Closed   * message from the TIP
@L 42    * user logs in to host
          .....
@C       * close connection to host
Open     * message from TIP
NEXT HOST NUMBER OR RING OFF NOW >
```

Figure 1—A Typical QUES Scenario

When the user closes his connection to the remote Host, QUES is once more able to connect to him and requests the number of the next Host required (name and password are not requested again). This procedure is then repeated continuously.

The only exception to this procedure is in the special case where the Host the user wishes to access is the National Library of Medicine (NLM). The British Library, as part of its Short Term Experimental Information Network Project (Ref. 8) has a number of centres which access the MEDLINE system on the NLM IBM 370/158. This machine is not connected to ARPANET as a conventional Host, but rather simulates five interactive terminals on the National Bureau of Standards (NBS) TIP.

Since this makes the process of connection extremely inconvenient and also provides little status information, we have written a program which also runs under our network software on the PDP-9 and which automates the connection procedure. This program will be described in detail elsewhere; however, it should be noted that, if a user specifies that he wishes to access the MEDLINE system (by giving the NBS-TIP number or MEDLINE when asked for Host number), he is automatically routed to this program. If this program is unable to make the connection (or if the user states that he does not wish to use the program by specifying the Host as NLM), QUES allows the user two minutes rather than the standard twenty seconds to make the connection since the procedure is considerably more complicated in this case.

All the data supplied by the user are printed by QUES onto a paper tape (thus obviating problems such as closing files after a system crash); similarly, when the user disconnects from his Host, his connect time is recorded on the tape.

In the unlikely event that the system does crash, it is convenient to know the time, and this may prove difficult at night when there is little activity. For this reason, QUES prints a message to indicate that it is still running every half-hour.

A sample of the output from the program is shown in Figure 2.

DESCRIPTION OF THE ANALYSIS PROGRAMS

A paper tape, as in the above section, is converted into a 360 job by the addition of a few lines of Job Control Language. This job is then sent to the RL machine via our other PDP-9.

Such tapes may have many errors. For example, due to system crashes, the "MONITORING TERMINATED" message may not have been printed; due to hardware problems, characters may be missing etc. Also, each interaction has generated two lines of out-

```
14 JUL 75 12 45 QUE: 1 **QUES MONITORING 6 PORTS FROM PORT£ 70
14 JUL 75 12 46 MED: 5 TIME MCH MLNS UCH ULNS @S "STO "PRI
14 JUL 75 13 05 QUE: 2 UNKNOWN HOST NUMBER
14 JUL 75 13 05 QUE: 724           QMC 87
14 JUL 75 13 05 QUE: 724           QMC 86
14 JUL 75 13 10 QUE: 74FORSEY      WESS 147
14 JUL 75 13 10 QUE: 74FORSEY      WESS 147
14 JUL 75 13 13 MED: 74 000106 0147 008 027 003 00 00 00
14 JUL 75 13 13 QUE: 74000105
14 JUL 75 13 14 QUE: 75FORSEY      WESS 147
14 JUL 75 13 14 MED: 75FORSEY      WESS 147
14 JUL 75 13 18 MED: 75 000130 0000 011 036 004 00 00 00
14 JUL 75 13 18 QUE: 75000129
14 JUL 75 13 18 QUE: 2 UNKNOWN HOST NUMBER
14 JUL 75 13 18 QUE: 75FORSEY      WESS 146
14 JUL 75 13 19 QUE: 75FORSEY      WESS 147
14 JUL 75 13 19 MED: 75FORSEY      WESS 147
14 JUL 75 13 21 QUE: 6 QUES OK
14 JUL 75 13 21 MED: 75 000033 0000 003 009 001 00 00 00
14 JUL 75 13 21 QUE: 75000032
14 JUL 75 13 21 QUE: 2 UNKNOWN HOST NUMBER
14 JUL 75 13 21 QUE: 75FORSEY      WESS %HME
14 JUL 75 13 21 QUE:2 UNKNOWN HOST NUMBER . . . ACCESS PROHIBITED
14 JUL 75 13 21 QUE: 75FORSEY      WESS %HME
14 JUL 75 13 52 QUE: 72003810
14 JUL 75 13 52 QUE: 724           QMC 66
14 JUL 75 13 57 QUE: 6 QUES OK
14 JUL 75 14 13 QUE: 72001652
14 JUL 75 14 18 QUE: 8 MONITORING TERMINATED
```

Figure 2—Sample PDP-9(B) log output

put, the first when QUES interrogated the user, the second when the user closed his connection to the remote Host.

Therefore, the first phase of analysis consists of a program called LOGB. This program checks the input for errors, removes superfluous lines (e.g., "QUES OK"), removes lines where the user has given an incorrect password or Host (these are printed out for inspection, but not passed onto the analysis programs) and compresses the data. In particular, the connect time is appended to the message generated when the user logs in. Two problems arise. The first is that monitoring may be terminated while a user is still logged in; in this case, the QUES program prints out the connect time up to the termination and so the time recorded is an underestimate. This of course presents no problems to LOGB, since there is no difference between such a message and a genuine log out. The second case, when the system crashes while users are logged in, does present a problem; in such a case, LOGB detects this by the occurrence of a "MONITORING STARTED" message not preceded by a "MONITORING TERMINATED" message. It generates the latter and the associated logout messages at the last time for which it had a valid message (hence the reason for the production of the "QUES OK" messages).

The program also produces messages indicating the number of ports in use. Due to a temporary restriction on the number of channels available in the PDP-9, QUES only monitors six ports at present. It is a simple matter for LOGB to record the number of ports in use except for one case, when QUES starts monitoring. In this case, ports may already be in use; the version of QUES to which we are referring did not attempt to distinguish whether the port was in use by a genuine user or not (e.g., someone having dialled the TIP number by mistake). This distinction can be deduced by QUES with a reasonable degree of certainty by knowing the TIP timeout period. This will be done in future, but the timeout period is not guaranteed to remain constant and the method is not completely reliable. In the data we present here this was not done, and LOGB had to deduce the number of ports in use. Due to the mode of operation of the telephone system, a user is allocated to the lowest numbered free port on dialling up. At present, we monitor ports 70 to 75 (octal). If the first port to be used after QUES was initialized was 71, LOGB would assume that port 70 was in use at that time. LOGB takes steps to ensure that such an error is not propagated if the port was not in genuine use.

Thus LOGB produces an output file, an example of which is given in Figure 3 (the output is that produced with the data in Figure 2 as input). This file is in a standard format and may be assumed to be error free. It is then passed to a set of analysis routines called XFSTATS for analysis. It was not expected that all the analysis functions which might at some time have been required could be specified in advance; therefore XFSTATS was written in a flexible, table-driven manner. The initial phase of the program consists of reading a file of control cards which are parsed. It then

```
14 07 75 12 45 Q£1
14 07 75 13 05 Q72 4                                  QMC     86  00  38  10
14 07 75 13 05 Q£9 03
14 07 75 13 10 Q74 FORSEY                             WESS 147  00  01  05
14 07 75 13 10 Q£9 04
14 07 75 13 13 Q£9 03
14 07 75 13 14 Q75 FORSEY                             WESS 147  00  01  29
14 07 75 13 14 Q£9 04
14 07 75 13 18 Q£9 03
14 07 75 13 19 Q75 FORSEY                             WESS 147  00  00  32
14 07 75 13 19 Q£9 04
14 07 75 13 21 Q£9 03
14 07 75 13 52 Q£9 02
14 07 75 13 52 Q72 4                                  QMC     66  00  16  52
14 07 75 13 52 Q£9 03
14 07 75 14 13 Q£9 02
14 07 75 14 18 Q£8
```

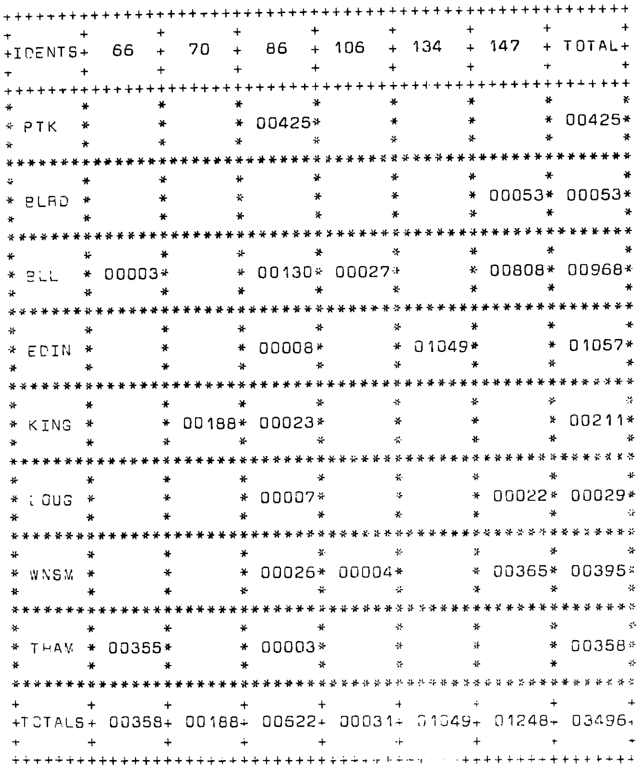
Figure 3—Sample output from the data reduction program LOGB

reads the file output by LOGB, selecting only data between the dates specified by the control file. These data are mapped into a structure in core, then, depending on the control cards, various analyses are performed, for example, a matrix of connect times for each host and user (see Figure 4).

RESULTS OF ANALYSES

In a paper such as this, it is neither possible nor desirable to give a full analysis of the results we have obtained (these are available in Reference 6), and we concentrate on three aspects of the results. In the following, the data to which we refer is, in the main, that obtained in July 1975. A single month provides a reasonable volume of data covering various times of day. July was chosen specifically since it was the month in which we did most monitoring; PDP-9B is also used for system development and monitoring is only performed when it is not used for that purpose. Thus the amount of monitoring varies considerably from week to week and much of the monitoring is at night and weekends with few users; at such times, the main value of the system is for controlling improper access. In the month concerned, we monitored the TIP for 525 hours (70 percent) broken down in the following way:

	0000- 0800	0800- 1300	1300- 1800	1800- 2400	Week- ends
Total (mins)	9120	5571	1713	4426	10674
Mins/day (av)	397	242	74	192	1334
% of time monitored	82.7	80.7	24.8	53.5	92.7



```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+      +      +      +      +      +      +      +      +
+IDENTS+  66  +  70  +  86  + 106  + 134  + 147  + TOTAL+
+      +      +      +      +      +      +      +      +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*      *      *      *      *      *      *      *      *
* PTK  *      *      * 00425*      *      *      * 00425*
*      *      *      *      *      *      *      *      *
****************************************************************
*      *      *      *      *      *      *      *      *
* BLRD *      *      *      *      *      * 00053* 00053*
*      *      *      *      *      *      *      *      *
****************************************************************
*      *      *      *      *      *      *      *      *
* BLL  * 00003*      * 00130* 00027*      * 00808* 00968*
*      *      *      *      *      *      *      *      *
****************************************************************
*      *      *      *      *      *      *      *      *
* EDIN *      *      * 00008*      * 01049*      * 01057*
*      *      *      *      *      *      *      *      *
****************************************************************
*      *      *      *      *      *      *      *      *
* KING *      * 00188* 00023*      *      *      * 00211*
*      *      *      *      *      *      *      *      *
****************************************************************
*      *      *      *      *      *      *      *      *
* LOUG *      *      * 00007*      *      * 00022* 00029*
*      *      *      *      *      *      *      *      *
****************************************************************
*      *      *      *      *      *      *      *      *
* WNSM *      *      * 00026* 00004*      * 00365* 00395*
*      *      *      *      *      *      *      *      *
****************************************************************
*      *      *      *      *      *      *      *      *
* THAM * 00355*      * 00003*      *      *      * 00358*
*      *      *      *      *      *      *      *      *
****************************************************************
+      +      +      +      +      +      +      +      +
+TOTALS+ 00358+ 00188+ 00622+ 00031+ 01049+ 01248+ 03496+
+      +      +      +      +      +      +      +      +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 4

The three aspects with which we will concern ourselves are:

(i) The global picture: overall usage and general statistics
(ii) The pattern of usage by one specific user
(iii) The pattern of usage of one specific host

On the global picture, we monitored the TIP for 31,504 minutes in the month. The total time users were connected to various Hosts was 10,761 user minutes, giving a usage of the six ports monitored of 5.7%. However, much of the time monitored was at weekends and night and the breakdown of this figure over the time periods used above is:

	0000- 0800	0800- 1300	1300- 1800	1800- 2400	Week- ends
Time*ports	94	8039	1443	462	723
% ports used	0.2	24.1	14.0	1.7	1.1
Logins	2	435	62	42	44

(where the percentage referred to is the percentage port utilization). The high figure in the 0800-1300 time is due to the non-availability of many US hosts to us in their prime shift (1300 onwards, UK time). This restriction applies to the two Hosts that were used most heavily, NLM and the Information Sciences Institute (IST) PDP-10X. A general matrix is produced of the usage at each Host by each user group. The complete matrix in our case would be 100×40 and would be quite unreadable. A partial matrix is shown in Figure 4. The two most heavily used Hosts during this period were Host 147 (64 percent) and ISI (9.4 percent). The reason that the former was used so heavily was that it had been unavailable for most of June; in addition, the general university usage was low in July because of the onset of holidays.

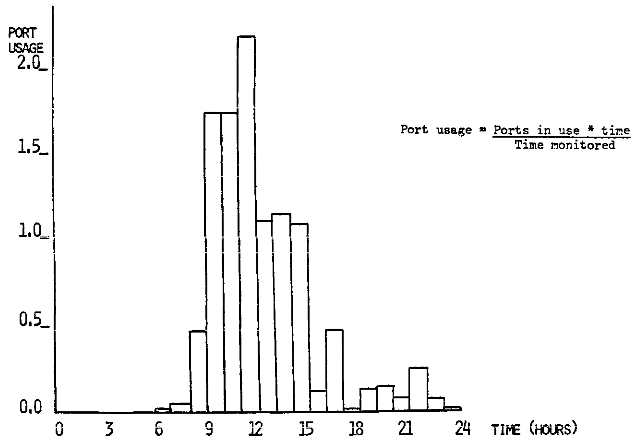


Figure 6

The pattern of usage overall is much as expected and is shown in Figure 5. In this, attempts to connect to a Host which was not available are excluded (QUES records this as a connect time of zero; there were 115 such occurrences in July). The second histogram shows the zero to ten minutes segment on a larger scale; the zero to thirty seconds time period contains many users who, although connected to the Host, were unable to log-in either due to the Host refusing to allow the log-in or rejecting it due to illegal account parameters.

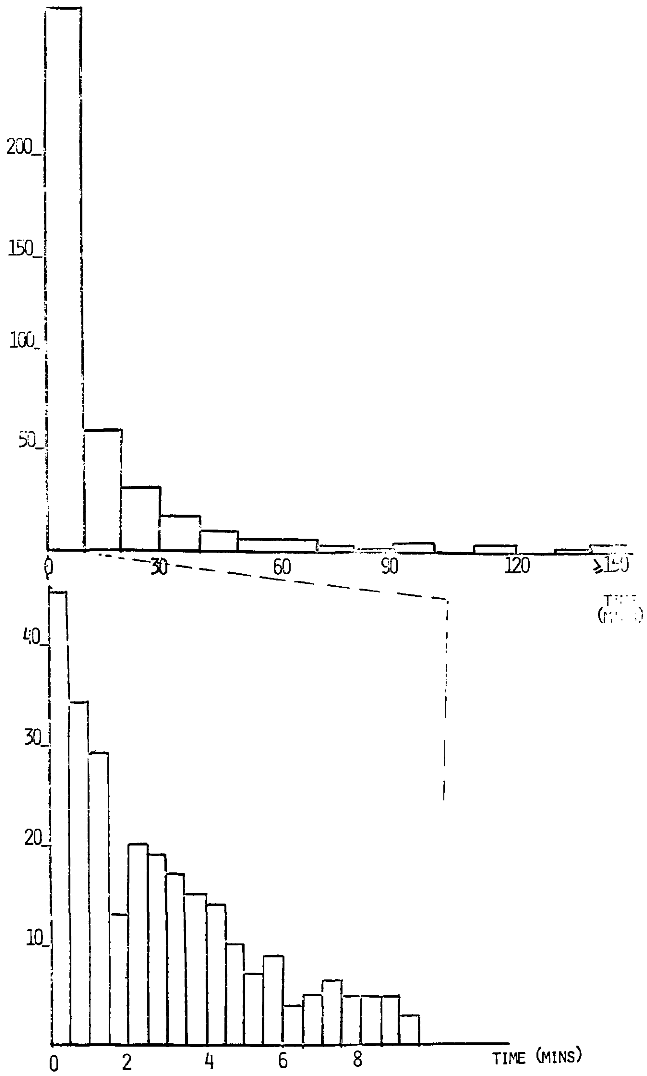


Figure 5

It is of considerable interest to determine the number of connections made and ports occupied and these are best normalized with respect to the average number of ports in use. Typical results of the ports in use over particular periods are shown in Figure 6. We also have histogram information on the number of ports in use at any one time. This information can be used to guide the TIP owner on the number of dial-up ports he should provide.

The second aspect which we wish to examine is the pattern of one specific user (Peter Kirstein). His usage this month was confined exclusively to one Host, ISI, and there were three types of usage; these are clearly reflected in the connect times (Figure 7). The first is

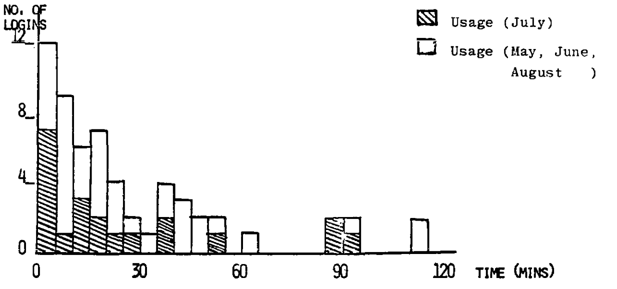


Figure 7

usage such as reading mail or sending messages. This takes a relatively short time, of the order of ten minutes. The second usage is editing documents and this gives rise to connect times of the order of half an hour. The third usage is entering documents and, to a greater extent, teleconferencing where connect times are of the order of hours. This pattern shows up clearly in the histogram of Figure 7 but, since there were only nineteen logins recorded in the month, we also give the corresponding histogram for the four month period May-August 1975 when there were 59 logins.

The last usage we consider is that of a specific Host. As we mentioned above, a number of centres in Britain access the MEDLINE system on the NLM IBM 370/158 to perform bibliographic searches. In addition to the QUES monitoring, we are able to monitor other details since each line from the user and from NLM passes through the PDP-9. In particular, we monitor the number of characters and lines each way. We also record the number of searches carried out, but this depends on the user specifying this number accurately to NLM; also, the actual number is, to a certain extent, subjective. Therefore, we do not make significant use of this information.

In the month we are considering, we recorded 148 logins to NLM, giving a total connect time of 7385 minutes. Eliminating those logins where MEDLINE was not available (shown by a very short connect time —of the order of a few minutes—and the user specifying that no searches were done and giving no "PRINT" commands), we had 87 logins.

Although the number of observations is reasonably high, the variation in the parameters was surprising. The greatest consistency was in the time spent logged in, an average of just under 33 minutes with a rms deviation of 11 percent of this value. Two parameters which might have been expected to be fairly consistent, the average number of characters per line to and from NLM each session, showed wide variations over the recorded sessions. The averages were 16.64 and 13.65 respectively with rms deviations of 31 percent and 49 percent of those figures. Similarly high deviations were shown in the ratio of the lines from NLM to lines from the user (52 percent), the time per line from NLM (46 percent) and the time per line from the user (58 percent). These results are tabulated in Figure 8.

The results we have presented above for the use of NLM are our initial results and will be supplemented with more detail when our current extensions to the monitoring system are completed. In this, all interactions with NLM may be copied onto magnetic tape and hence a complete analysis may be performed. These data will be used partially to check the data generated by some of the users under the project of Reference 8. They will be of particular use in providing a quantitative basis for certain subjective criteria, for example, response times. It is hoped, at a later date, to extend this system so that it may be used to monitor the interactions with any Host on ARPANET.

	Total	Average	RMS Deviation
Time logged in	171 349	1969.53	216.83
Characters from NLM	551 925	6343.97	5318.70
Lines from NLM	32 569	374.36	284.04
Characters from user	105 979	1218.15	2087.59
Lines from user	8 554	98.32	171.23
Average time per line from NLM	—	5.66	2.57
Time per line from user	—	42.01	24.26
Characters per line from NLM	—	16.64	5.16
Characters per line from user	—	13.65	6.65
Lines from NLM/Lines from user	—	7.86	4.05

Figure 8—Characteristics of NLM usage (times in seconds)

CONCLUSIONS

The method of monitoring and access control we have developed is not of general application, particularly since it requires a dedicated processor in addition to the one being monitored and, in the general case, it is obviously simpler to put these functions in the latter. However, the introduction of monitoring and/or access controls into a computer brings with it a loss of reliability (since, not only does it increase the complexity of the software but also it requires additional hardware). This loss of reliability may not be acceptable and so our technique may be of more general applicability.

An example of this is evinced by ARPANET. Access controls were introduced into the TIPs in late 1974 and, for a number of reasons (in particular, the decrease of reliability that backing store would introduce and the problems of maintaining the password data base) it was introduced in a way that led to inconvenience for users and large network overheads. Although our approach would not obviate the problem of maintaining the data base, it would certainly not decrease the reliability of the TIPs. At the present time, the LONDON-TIP is the only node on ARPANET on which monitoring of users and access control is being carried out.

By use of this technique, we have obtained considerable monitoring data and have used this to explore the methods of usage of the network. We are currently developing other means of monitoring in conjunction with QUES to enable us to obtain a more complete view of the usage of our node. The above types of figures are useful in giving a general overview of the extent of usage of the network for different applications. The measurements give an excellent cross-check on subjective reports from user groups on their usage of specific Hosts (Reference 9). Over a period of time, we expect these measurements to be extended also to leased line ports and so to give us a complete picture of the usage of ARPANET via the UCL node.

ACKNOWLEDGMENTS

We wish to acknowledge support by the British Library under grant SI/G/093. Also, such research would not be feasible without support of the many bodies for the ARPA project at University College, in particular, the Science Research Council (B/RG/5981), the US Advanced Research Projects Agency (N00014-74-C-0240) and the UK Ministry of Defence (AT/2047/064).

We also wish to acknowledge the contributions of many people of the INDRA group at University College to this project, in particular, to Roger Gould and Peter Mott who did much of the coding and debugging of the programs mentioned.

REFERENCES

1. Roberts, L. G. and B. D. Wessler, "The ARPA Network," *Computer Communication Networks*, Prentice-Hall, pp. 485-499, 1973.
2. Kleinrock, L. and W. Naylor, "On Measured Behaviour of the ARPA Network," *Proc. AFIPS NCC*, Vol. 43, May 1974, pp. 767-780.
3. Opderbeck, H. and L. Kleinrock, "The Influence of Control Procedures on the Performance of Packet-Switched Networks," *Proc. Nat. Telecomm. Conf.*, San Diego, Dec. 1974. pp. 810-817.
4. Bolt Beranek and Newman, Quarterly Technical Report No. 2 (1 April—3 June 1975), Report No. 3106, July 1975.
5. Ornstein, S. M. et al, "The Terminal IMP for the ARPA Computer Network," *Proc. AFIPS SJCC*, Vol. 40, May 1972, pp. 243-254.
6. Stokes, A. V., *Analysis of Usage of the British Node of ARPANET, July-December 1975*, TR-30, Department of Statistics and Computer Science, University College London, 1976.
7. Stokes, A. V. and P. L. Higginson, "The Problems of Connecting Hosts into ARPANET," *Proc. Eur. Conf. Comm. Networks*, pp. 25-33, 1975.
8. Holmes, P. L., *An Approach to the Development of Library and Information Networks with Special Reference to the United Kingdom*, AGARD-CPP-158, p. 6-1, 1974.
9. Kirstein, P. T., "Distributed Computer Networks," *Nature*, Vol. 257, Oct. 1975, pp. 549-554.

Office automation project—A research perspective

by HOWARD LEE MORGAN
University of Pennsylvania
Philadelphia, Pennsylvania

ABSTRACT

This paper attempts to place some perspective on the research and developments going on in office automation. It describes the functions which can be assisted by computers, and indicates where more research may be needed. A brief description of the Office Automation Project at the Wharton School is provided. The systems being developed include word processing, electronic mail, decision aiding technology, and integration with various databases. This effort is compared with some of the other, complementary research projects in office automation under way around the country.

INTRODUCTION

Having successfully occupied the accounting and financial centers of business for almost two decades, and the production control centers for one decade, the computer industry is finally beginning its march on the office. Until now, this was the last stronghold of human activity in many of our large business and government organizations. Office automation, of which word processing is but one element, is one of the fastest growing segments of the marketplace. A large number of companies have entered the field, and sales of word processing equipment alone should reach $.75 billion in calendar 1976.[10]

This paper is an attempt to place some perspective on the research efforts going on in office automation, with some emphasis on structuring the many different services which can come under this heading. It also tries to indicate where there are problems of research rather than development interest, with the hope that this may stimulate various groups to begin work on them. Finally, it describes the efforts under way at The Wharton School in this area, the Office Automation Project(OAP).

ASSUMPTIONS

This author has based his research on the following assumptions. These are not all universally accepted, although I have found agreement on some of them in each of the major research efforts studied.

1. More efficient production of paper is *not* the ultimate goal of office automation. While a side benefit may be that documents, letters, etc., can be produced and changed more efficiently, one hopes to eliminate more paper than is produced in order to attain a truly automated office.
2. The burden of proof is on those making changes. Whenever we propose to make radical changes in the way in which offices function, we should be prepared to justify these changes. Technology itself is not sufficient reason to introduce change. A corollary to this assumption is that in order to have the change accepted, there must be some short term payoff to those who must use the system (managers and clerical workers).
3. Office problems are not well structured. The reason that computers have been able to take over accounting and financial departments is that the problems there are relatively well understood and structured. The type of activity and decision making which characterizes most offices does not lend itself to such clear structuring, and thus demands different solutions.
4. Both technological and organizational research challenges remain. Because of the semi-structured nature of office problems, new data structures, program structures and even hardware may have to be designed to really attack the problems. Also, since telecommunications may permit automated offices to exist in quite a different physical manner than at present (e.g., offices in the home), the impact on organizational designs and strategies require much research.

These assumptions have led us to the particular analysis and development approaches used in my contributions to the Wharton Office Automation Project.

FUNCTIONS

Top down methodology is much in vogue these days, for its supposed clarity and completeness. Here, a top down analysis would begin with the functions performed and the people performing them. We should note that the actual system development and implementation at Wharton is being done in Ness' "middle out" approach,[16] which tends to lead to greater user involvement and short term payoffs.

Since office functions are in some sense dependent on the particular business, we must describe the environment in which we work. The impetus for the Wharton project came with the formation of the new Department of Decision Sciences at The Wharton School. This consists of 15 teaching research faculty, about 25 support personnel (secretaries, research assistants, etc.) and has a consolidated budget of slightly under $1 million/year. There are several middle managers (typically principal investigators on projects) and a department chairman, who may be considered a more senior manager, since he must worry about the consolidated budgets from teaching and research grants. We suggest that our office is therefore similar to offices in information oriented companies, e.g., banking, insurance, R&D, rather than product oriented companies. We are aiming at aiding the middle manager in performing his or her functions. These functions fall into six major classes. For each, we discuss what has been done by the industry to automate it, and what might be done.

Communications

Mintzberg, in his study of managerial functions, reports that almost all of most managers' time was spent on communication.[14] This is borne out by the fact that it is in this area that the most work on office automation has proceeded. We divide communications into a number of categories for purposes of discussion.

1. formal/informal
2. reply required/no reply required
3. message/document (short/long)
4. internal/external
5. voice/text/graphics

The word processing industry has realized the amount of time and money spent in the average office on communications and has attempted to automate some of these tasks. They have had much success with long, formal texts, as typified by legal contracts, large reports, and mass letter writing examples. The main technology involves text editors, about which we shall have more to say later.

The informal message area has proven of tremendous value to those organizations that have implemented systems such as electronic mail. In one large New York bank, for example, the time to get messages and memos from Wall Street to Midtown was cut from four hours to five minutes. The ARPANET community has also noted the powerful sense of community which the mail facility can create.[5]

In the Wharton system, features for extracting replies are a part of the electronic mail system, but are just beginning to be used. The phone or voice communications area is untapped. Services such as logging calls, handling of phone messages, etc., could be provided. Some recent research,[2] has indicated that for office type of problem solving, voice may be more effective than face to face communications, even given the lack of graphics.

One West Coast research group is working heavily in the area of computer graphics integrated into a word processing system, and the Binary Image Processor created to store and retrieve technical manuals also is attacking this area.[15]

Information storage and retrieval

Almost all of us keep files of one sort or another in our offices. Names and addresses, correspondence, task related work, all seem to accumulate in our file drawers. An informal survey of the faculty members in the Department reveals that the middle managers have on the order of 1,000 files, and the chairman has about 2,000 separate file folders. Most of us seem to be able to index the information in these files in our heads, or with simple filing schemes. The research literature on information storage and retrieval tends to lead to strategies which may be more rigid than those we use when searching our own files. For example, I may store a letter in a folder with the recipient's name on it, or in a folder with the name of a paper he or she has written, or in some other task related file. Yet I can usually retrieve all correspondence to a person by remembering the subjects of the letters.

There are basically three problems to effective storage and retrieval. First, storage of full texts is expensive. For the number of times which I typically reexamine a letter, it would be hard to justify online storage, even with the relatively cheap terabit memories. This can be alleviated by storing only the index online, perhaps with abstracts, such as is done for document collections.[21]

Second, there is the problem of data input. Letters or documents prepared on an automated system can be easily stored. However, material received on paper from the outside incurs additional costs for input. As the use of electronic mail increases, this problem too may be overcome.

Finally, there is the problem of indexing. Automatic indexing of the type done in library systems[22] is content oriented. The degree of success of the automatic methods depends greatly on the threshold values in the algorithms. Short letters are quite hard to index, because few of the terms occur frequently enough to exceed thresholds.

One research group has suggested that the best aid would be to simply show the user a large number of file folder headers, much as one would see them as one opened the drawer, and let the human pattern recognition and search process be used to select from that set. Depending on the hardware, one could show from 50 to several hundred such folder names simultaneously. This scheme should work for the few thousands of files discussed so far. If we wish to integrate several people's files, this approach is not the one to follow.

Data analysis

Tools for analyzing numeric data have been available and in use by sophisticated managers for years. Regression, statistical packages, forecasting models, etc., have all been well developed to suit this market. Analysis of the other types of data which a manager sees, such as newspapers, magazines, reports, etc., is still beyond present day capabilities.

The Very Large Database (VLDB) project being sponsored by ARPA,[7] is attempting to examine some of the problems of analyzing this "intelligence." For example, we may wish to examine all of the data linking our company and research in computer databases. This necessitates bibliographic searches, calls to friends in other universities, etc. Learning to use even the available systems (NY Times Index, MEDLINE, etc.) is hard enough without trying to integrate these and automate access to them.

Decision making

Most middle managers are not making repetitive, routine decisions but rather are making semi-structured decisions which recur on an irregular basis. These tend to be tactical rather than strategic decisions. As part of an ONR sponsored effort we are building DAISY,[13] a decision aiding information system. This attempts to automate the memory of an organization to support a manager making decisions, while still giving the manager the freedom to make the decision. Some examples of such semi-structured decisions are choosing an acquisition partner, deciding upon a battle plan, or trying to allocate fire fighting units in a municipality.

A budget planning process has been implemented and partly integrated with the OAP and DAISY by Purves and Godard.[20] With this, a professor can learn how to budget for a research proposal, go through the calculations with a program, and dump the results in text format into the budget pages of a formal proposal. We are working with other decision processes, but are excited about the potential for linking this type of system to both the communications and the information storage and retrieval functions of the OAP.

Personal assistant

Goldstein[9] has described an artificial intelligence system for aiding in preparing schedules. This type of activity is categorized as a personal assistant. It understands some portion of the manager's decision rules, priorities, and requirements, and attempts to maintain schedules. In the Wharton OAP, Ness has implemented a simple scheduling system which permits a person to store and retrieve his or her calendar for any day or group of days. It also automates the reminder function, by printing, in priority order, items which users have placed on lists (e.g., Write letter to x, Buy wife's birthday present, etc.).

The short term payoff and appeal of a personal assistant is quite high. It tends to draw less computer-oriented people into using the system. Other possible personal assistants might include arranging travel reservations, maintaining working paper distribution lists, and choosing appropriate referees for journal papers.

In the long run, such applications of Artificial Intelligence and knowledge based systems[12] will be routine. They offer a great potential area for research in the fertile office automation applications area.

Linkage to corporate databases

Under communications, we discussed the interactions between people. Managers and other office personnel also routinely examine and update various corporate databases. These databases are maintained by and under the control of other offices, typically the data processing department.

It is desirable to permit the office automation system to have direct access to these databases. For example, the budget planning system described under decision making should not only feed the text processing parts of the office system, but should be able to create the proper entries in the accounting database to create the new project which has been proposed and budgeted. Other examples of such linkages might include moving name and address information from personal mailing lists into corporate customer databases, and examining personnel databases for skills required on particular projects.

The key research problem posed by providing this function is that of designing a simple query language for the databases. This is being examined at the present time by the End User Facilities Task Group of the CODASYL Committee.[4]

PRODUCTS

The functions described above have led to the development of a number of products in attempts to automate one or more of these. First and foremost are *text editors*. These have been around for a long time and are the backbone of most timesharing systems. Unfortunately, few of these line oriented editors are very good for office automation. Several major research groups (SRI-ARC, Xerox, USC-ISI) have all realized that good editors require separation of the editing commands from the text being edited. This can be done on CRTs, but not on the typewriter like devices which most timesharing systems have to support. The NLS system[1] is an excellent example of sophisticated editing. This system also changes the way in which people think about documents, emphasizing the hierarchical nature of long documents, and the ability to view a document at various levels of detail, i.e., chapter headings, section headings, etc., down to sentences and paragraphs.

The sophisticated editors at another installation not only separate command and text being edited, but also always try to show the user final formatted copy, even if this involves multiple fonts. The VyDec word processing system also works in this mode, which is quite appealing to secretaries and other initial document entry personnel. Since they can see an attractive, finished form as they enter a draft, a reward is present, reinforcing their desire to use the system.

One conclusion that can be drawn from studying the best editors available is that they require much more in the way of CPU and memory resources than most people estimate. To present an 8.5 by 11 inch screen with several type fonts may require one million bits of storage for the image alone.

Electronic mail packages are provided on ARPANET and have been developed in house by a number of companies[8] and timesharing bureaus. Except for the Wharton system, most of these seem to treat all mail the way the US Postal Service does, i.e., as first class mail, without any real indication of the type or content.

Integrated word processors are machines combining an editor with some minicomputer and storage devices. These go under a number of names, but are really automated typewriters. Unfortunately, most of them lack communications ability, and hence could not serve for the full range of functions mentioned in the FUNCTIONS section.

Database managers for handling the semi-structured types of information used in organizations are available. Many firms market name and address list maintenance programs, and the general database management systems can easily be used for these purposes. It is important that these systems permit easy addition of fields, storing of textual information, and extraction from files.

These are just some of the products under development or available for commercial use. We next describe briefly the first product of the Wharton OAP, the Office Automation System (OASYS).

OASYS

Figure 1 is a block diagram of the major functional pieces of the Wharton Office Automation System, designed and developed for the PDP-10 in MACRO by Professor David Ness. Those marked with asterisks are being reprogrammed or added by the author and others, using higher level languages, and DBTG database technology.

The text processing and runoff systems are a set of extensions to the DEC Runoff program for producing formatted text. Some of the more interesting features created by Ness are in the personalization, history and profile areas.[19] Each of the programs in OASYS is designed to deal with several people. The "for" user is the manager for whom the program is being used. The "by" user is the person who is actually at the terminal, e.g., a use of the schedule program *by* a clerical worker *for* a manager. The profiles tell these programs what input forms the *by* user desires depending on level of skill, etc., and what output forms the *for* user demands. The history features permit the system to inform each user when any of the questions or commands have changed, or when new features are added to the system. This is done by recording for each user, module, and command, a level number. Before the system asks a question, it determines whether or not the user has seen this version. If not, it is prefixed by (NEW).

The profile permits users to further personalize the modules by specifying a user/module combination which questions or prompts may be skipped. Thus, in creating name and address files I am asked for home address, which I routinely keep, but Professor Ness is only asked for business address.

The electronic mail system has several types of mail. A user is told how much of each class is waiting for him or her at login time. Currently, the system permits us to distinguish among BUG mail, relating to system bugs, regular mail, return receipt mail, which auto-

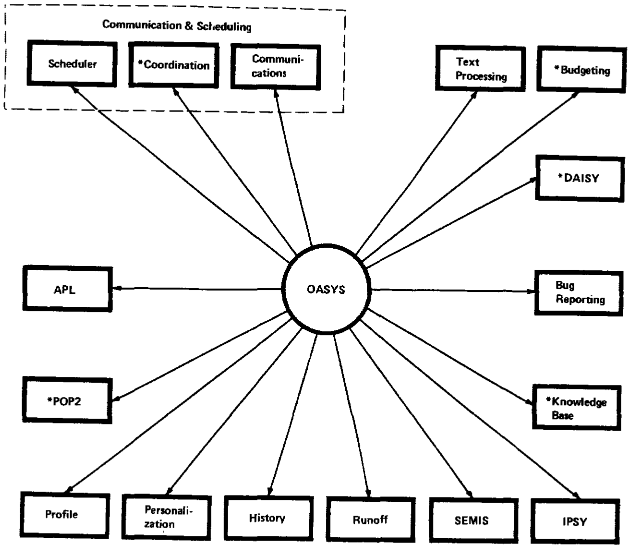


Figure 1

matically sends a receipt to the sender when it is examined, control mail, which deals with system operation, and LOG mail, which deals with account usage. In addition, regular mail is prefixed with an indication of the mailing list used to send it, or (PERSONAL), if it was individually addressed.

Users report bugs by specifying the module with the bug and writing a message. The system sends this to a responsible party, noting that users have reported bugs. Users are automatically notified through the mail system when the staff has received the bug report and when they have disposed of it.

Working in the POP[2] language, the author and Mike Zisman have recently integrated processing of databases, letter writing, and complex processing for handling manuscripts being considered for publication in professional journals. This includes selecting referees for papers and writing the appropriate correspondence for the referees and authors from the editors.

More information on OASYS is available in de Blasis[3] and in Ness.[17, 18]

TECHNOLOGIES

One of the key questions in office automation is the type of hardware and software technology which should be used for developing products. Wharton and SRI-ARC have both chosen to work on large central processors with communications facilities and relatively unintelligent terminals. Others have chosen more distributed technology, emphasizing minicomputers and some communications ability. This author feels that the latter route is the one likely to triumph in the medium term, but that we are likely to see a relatively large minicomputer supporting a cluster of terminals in one office, rather than one at each terminal. All of these minicomputers would, of course, be connected together for mail and database access.

As noted under products, some form of video display terminal (CRT, plasma panel, etc.) which can handle full graphics is required for the long term solution. It is not clear whether or not we should retain the present aspect ratio dictated mostly by the availability of cheap television (525 line scan) devices. In any case, the speed with which the display communicates with the machine for output should be as high as possible. Graphical input devices are also required, and the "mouse,"[6] a device which is rolled around on a hard surface and tracked by a cursor on the display, seems to be an attractive current answer.

On the software side, systems have been developed in languages ranging from assembly to INTERLISP. One would hope that higher level languages would be used more extensively, but since editing is often character oriented, reasonable efficiency can be achieved on many machines only by staying close to assembly level. Languages such as BCPL and BLISS permit this, while still retaining advantages of high level languages.

Sorting is such a fundamental operation that it must be considered a part of the software technology. In a similar vein, good man/machine dialogues are needed for any system which is going to have unsophisticated users. Martin[11] describes these techniques in quite a bit of detail.

RESEARCH AREAS

While much has and continues to be done in the office automation area, I have identified three major technological and two major organizational areas which are ripe for research. Some of these areas are beginning to be examined by members of the OAP at Wharton.

1. Input processing. Almost all of the functions which have been automated involve the production of output. How can we efficiently handle input in automated ways? Some OCR technology, some automatic indexing technology, and man/machine interaction technology must be brought to bear on this problem. Even if we begin to receive a large portion of our mail electronically (and several ARPANET users now get more than 50% of their mail through the network), the time and difficulty of entering externally supplied information must be cut down.

2. Integration of databases. Integrating databases more complex than names and addresses is not available in most office automation systems. How to present a clear and simple interface to the end user is the major problem here.

3. Using artificial intelligence methods. We have begun work on a knowledge based system to assist me in processing papers for the journals on whose editorial boards I serve. This will "understand" the types of interactions I have with authors, referees, and other editors, and will try to generate the proper correspondence, database updates (i.e., correspondence logging), and action requests. There are many other places where AI can contribute.

4. Amount of personal communications needed. Organizational studies on the effectiveness of dispersing the office personnel to separate locations, communicating only through the system, must be made. Chapanis[2] has done some work on problem solving with various communication modes, but we really know very little about the best ways to do this. Should the people meet face to face once a week? Once a month?

5. Who gets the terminals. For years people in the MIS field have seesawed on whether or not the managers or their staff assistants will actually have the terminals on their desks. Someone should attempt to answer this question through controlled experiments.

SUMMARY

Office automation is a growing area of concern for computer and organizational researchers. The efforts to date have focussed mainly on word processing, which is only a part of one of the office functions. Much more attention needs to be directed to the other, higher impact, less structured activities which take place in offices. This should yield a set of high quality products which will change the way in which offices are run over the next two decades.

The Wharton Office Automation Project is attempting to attack some of the non-word processing areas, paying particular attention to database integration, good user interactions, and handling of decision processes. We realize that our comparative advantage is not in the hardware area, and hence are trying to build modules of a system which can be grafted onto good hardware as it develops.

It is only a matter of time before most of us will be interacting with other people with the aid of computers as an everyday occurrence. We owe it to ourselves and the field to ensure that this happens in an efficient and socially responsible manner. This paper has not discussed the obvious privacy/security requirements which will be necessary in office automation systems, but they must be thought of at the beginning of any design projects.

This paper has attempted to set down something of the state of research in the area in late 1975. Functions, products, and technologies have been characterized with an eye towards possible research topics. The author hopes that those who work on these topics will communicate with him.

ACKNOWLEDGMENTS

The Office Automation Project was the result of necessity and a number of discussions among Professors David Ness, E. Gerald Hurst, Jr., Thomas Johnson, Rob Gerritsen, and myself. Dave Ness has spearheaded the effort and did almost all of the programming for the first generation OASYS. The author has benefited from the above individuals' comments, as well as the recent programming and intellectual support of O. Peter Buneman and Michael Zisman, and the editorial assistance of Dorna Caskie.

REFERENCES

1. Augmentation Research Center, *NLS-8 Glossary,* Stanford Research Institute, Menlo Park, CA, July 1975.
2. Chapanis, A., R. Ochsman, R. Parrish and G. Weeks. "Studies in Interactive Communication: I. The Effects of Four Communications Modes on the Behavior of Teams During Cooperative Problem Solving," *Human Factors,* 1972, pp. 487-509.
3. de Blasis, Jean-Paul, "An Interactive System for Office Automation: Some Organizational Implications," *IRIA Colloques* January 14-16, 1976. (Available from Institut de Recherche d'Informatique et d'automatique, BP5—Rocquencourt 78150 Le Chesnay France).
4. Ledgard, H. (ed.), "Report of the CODASYL End User Facilities Task Group," *FDT 7,* March 1976.
5. Englebart, Douglas, Richard Watson and James Norton, "The Augmented Knowledge Workshop," *Proceedings of the National Computer Conference Vol. 42,* June 1973, pp. 9-21.
6. Engelbart, Douglas C., "Design Considerations for Knowledge Workshop Terminals," *Proceedings of the National Computer Conference 42,* June 1973, pp. 221-227.
7. Gaines, R. S. and R. Gammill. *Very Large Data Bases: An Emerging Research Area,* Informal working paper, RAND Corporation, Santa Monica, CA.
8. Galbraith, Jay, *Electronic Mail Systems,* Unpublished report, The Wharton School, December 1975.
9. Goldstein, Ira, "Bargaining Between Goals," *Proceedings of the Fourth International Conference on Artificial Intelligence* Tblisi, USSR, August, 1975.
10. Johnson, Tom, *Word Processing,* Working Paper #76-02-01, Department of Decision Sciences, The Wharton School.
11. Martin, James L., *Design of Man-Computer Dialogues.* Prentice-Hall: Englewood Cliffs, NJ 1974.
12. Martin, William, "Interactive Systems—Theories of Implementation," *Proceedings of the Wharton/ONR Conference on Interactive Decision Support Systems* (to appear) 1976.
13. Morgan, H. L., "DAISY: An Applications Perspective," *Proceedings of The Wharton/ONR Conference on Decision Support Systems* (to appear) 1976.
14. Mintzberg, Henry, "The Manager's Job: Folklore and Fact," *Harvard Business Review 53,* 4, July August 1975, pp. 49-61.
15. Ness, David, private communication, 1974.
16. Ness, David, "Interactive Systems—Theories of Design," *Proceedings of the Wharton/ONR Conference on Interactive Decision Support Systems,* (to appear), 1976.
17. Ness, David, *Office Automation Project: Text Processing,* Working Paper #75-05-02, Department of Decision Sciences, The Wharton School.
18. Ness, David, *Office Automation Project: Overview,* Working Paper #75-05-03, Department of Decision Sciences, The Wharton School.
19. Ness, David, *Office Automation Project: Personalization and History Features,* Working Paper #76-02-10, Department of Decision Sciences, The Wharton School.
20. Purves, Arthur and M. Godard, *An Interactive Budget Planning System,* Internal Report, Department of Decision Sciences, The Wharton School, Fall 1975.
21. Salton, G., *Automatic Information Organization and Retrieval,* McGraw-Hill, 1968.
22. Salton, G., A. Wong and C. Yang, "A Vector Space Model for Automatic Indexing," *Communications of the ACM 18* (November 1975) pp. 613-620.

Evaluating the impact of office automation on top management communication

by JAMES H. CARLISLE
University of Southern California
Los Angeles, California

ABSTRACT

This paper is concerned with the impact of new communication technologies on the effectiveness of top management decisionmakers. Word processing technology is only the beginning of a revolution in office automation and managerial communication which will include teleconferencing, electronic mail, and wide availability of personal computer-based systems. The potential problems and benefits must be considered in the context of the overall communication system and management needs of an organization. A research methodology is described which leads to the development of organizational models within which: (1) management communication problems can be anticipated, (2) solutions can be proposed and compared, (3) specific office automation systems can be designed, and (4) the impact of alternative systems on organizational effectiveness can be predicted and evaluated.

INTRODUCTION

Office automation is one of the new buzzwords used to describe the computer augmentation of day-to-day office functions. Most of these functions involve some aspect of the increasing volume of interpersonal and formal communications which take place within organizations. Facsimile, electronic mail, word processing, teleconferencing, on-line calendars, information storage and retrieval and general management information systems are available today to top management and even to middle management in many organizations. The combination of these (and more) functions into an integrated, computer-based system for use in managing organizations is likely to bring about an organizational revolution for white collar workers comparable in magnitude to that resulting from the introduction of the assembly line to blue collar work.

Anticipation of the organizational impact of this new communication technology is essential to the long range planning of both system developers and eventual user organizations. There is currently a lack of any generally accepted theory and methodology for evaluating the impact of today's office automation systems. Between now and the time when computer terminals and digital voice coder boxes sit on every manager's desk, there is an opportunity to study and improve the effectiveness of such systems. We need to study how and why the early systems are being used. More importantly, we need to understand how these systems impact the overall communication system of an organization and its management.

This paper describes the design of a scientific analysis of top management communication. This analysis results in a model of communication activity for the particular organization and manager studied. This model then supports a strategic analysis of management problems arising out of increased demands for communication and the evaluation of new communication technologies coming onto the market to "solve" those problems.

The orientation of the research described in this paper is top management effectiveness, not computer technology. Office automation should not be studied as an end in itself, but as a means to improved managerial performance. Top management stands to gain or lose the most from this new technology. Their valuable time and energy is largely devoted to communication activity of one sort or another. Freeing up time for top management and giving them increased potential for effective and rapid communication within and among organizations are attractive payoffs. On the other hand, if office automation is developed and introduced incorrectly, it could disrupt office communications, making them even less efficient, and create serious resistance to all of the new communication technologies. If office automation fails to gain acceptance and demonstrate value in the executive suite, it is unlikely to receive adequate financial support to be properly used throughout the organization and throughout society.

THE NEED FOR OFFICE AUTOMATION

One of the most significant problems facing management in the coming years is the rapidly increasing complexity of the information environment in which both long range and day-to-day decisions must be made. In his article on "The Future-Oriented Corporation," Dr. Burt Nanus observed that

> *We live in an age of increasing complexities, an age of macrosystems in which everything of importance that occurs anywhere in the world is immediately known everywhere else, thereby precipitating consequences which, in their own turn, provoke still other changes. The recent energy crisis is an excellent example of the interrelationship of political, economic, and technological factors and the enormous significance for corporate decision making.*[1]

So far these crises have been few and temporary in nature. Structured management information systems have, unfortunately, proved inadequate to supply the information needed. Much of the communication in such crises is unstructured, informal, and contains subjective and up-to-the-minute data. Today's organizations, with their traditional methods of processing letters and reports and their reliance on telephone, telex, mail and personal meetings for informal communications are ill-equipped for the volume of critical information flow during even a minor crisis. The problem is already with us, since past crises have been merely amplifications of the normal day-to-day flow of structured and unstructured communication within our increasingly interconnected communications environment.

Few managers escape the daily avalanche of unstructured communications from both inside and outside their organization. Fewer still have a clear enough model of their communication environment to support efficient and effective utilization of their time through scheduling and monitoring selectively those communications with the highest priority. There are secretaries and managerial assistants who perform this function quite well. The drawbacks to this solution are that it is becoming increasingly expensive and it leaves the manager highly dependent on that other person.

POTENTIAL PROBLEMS IN IMPLEMENTATION OF OFFICE AUTOMATION

The realization that we are rapidly entering an era in which information and communications are the limiting factors in performance of a majority of organizations is reflected in the recent excitement over office automation. Corporations are reaching for the elusive carrot of "cost reductions in word processing"—often irrationally and often with little understanding of the far reaching effects that the new communication technologies are having on their users. Most automated office projects up to this time have been fiascos.[2] Major suppliers of the hardware and software for "word processing" seem to know almost nothing about the informal organizational environment into which their systems have brought chaos. Not only have customers had serious organizational problems, but even the vendors themselves have had trouble with inhouse implementations of their text-editing and document preparation systems. The computer industry vendors seem to think that management communications can be improved just by speeding up the production of error-free text.[3,4] It's important to keep in mind that the "word processing" aspect of office automation is to communication automation what keypunching was to management information systems—only the tip of the iceberg.

There is a strong possibility that all the mistakes of the management information system era of the '60s will be repeated with office automation due to the zealous marketing of computer companies and management consultant firms hoping to jump on the bandwagon of this new communication technology.

One of the reasons which has been most frequently pointed out for the failure of management information systems and management science in general has been the lack of understanding on the part of top management itself of the theories on which these decision support capabilities were based. A second reason is the lack of understanding on the part of the system developers of how people within the organization actually do their work. Too many computer-based systems have already been designed on the basis of technological breakthroughs and innovations which were insensitive to the limit on man's rationality and the social needs that must be satisfied within organizational structures.

It is as if Russell Ackoff's plea to avoid the development of "Management Misinformation Systems" was ignored by computer system designers.[5] It is as if Chris Argyris' analysis of "Management Information Systems: The Challenge to Rationality and Emotionality," was understood only by those managers who had personally suffered the neglect or outright sabotage of management databases.[6,7] The computer industry has not responded adequately to the real challenges of providing useable and responsive management decision support systems. We are now at a point in time where,

> *The managements of using organizations are becoming hardened to the computer as they see frequently unfulfilled promises on the part of the vendors and their internal staffs. The net effect is a diminished esteem and an increased skepticism about the potential of the computer.*[8]

Despite the skepticism, which is probably only really a reflection of despair, the automated office is on the way. A U.S. Department of Labor report in 1970, describing patterns of U.S. economic growth, indicates that office computing and accounting machines are the fastest growing industry in the United States, with an average annual growth rate of over 10 percent.[9] Recent introduction of digital data networks such as the ARPA network, and value added networks such as TELENET, offer new telecommunication services which are highly competitive with telex, telephone, and the post office, as well as a viable alternative to at least some business transportation.

WHY IS RESEARCH ON MANAGEMENT COMMUNICATION NECESSARY?

The question of paramount importance is how do we get to the office of the future without destroying the social fabric of today's organization and without further dehumanizing communication within our society? To answer this question we must have a better understanding of the office of today, based on scientific examination of management behavior. The needs for any new communication technology should then be derived from analysis of that behavior and consideration of personal and organizational values.

Most of what little we do know of managerial behavior has been learned with primary concern for leadership style, rationality of decision-making and determinants of satisfaction and motivation. The communication needs and behavior of management are only a recent focus of investigation. Henry Mintzberg, who conducted one of the most extensive investigations of managerial work, recently noted that

> *I was struck during my study by the fact that the executives I was observing—all very competent by any standard—are fundamentally indistinguishable from their counterparts of a hundred years ago (or a thousand years ago, for that matter.) The information they need differs, but they seek it in the same way —by word of mouth. Their decisions concern modern technology, but the procedures they use to make them are the same as the procedures of the nineteenth-century manager. Even the computer, so important for the specialized work of the organization, has apparently had no influence on the work procedures of general managers. In fact, the manager is in a kind of loop, with increasingly heavy work pressures but no aid forthcoming from management science.*[10]

With the exception of Mintzberg's study, which proposed and supported an intriguing theory of managerial work roles, most of the research on management communication has attempted to measure attitudes and preferences among alternative communication media. This latter research has been supported or conducted primarily by the telephone companies of the U.S., Canada and Great Britain as a form of market analysis for picturephones. It should be noted that attitudes are not always correlated with behavior. Even behavioral studies with college sophomores conducted in research laboratories offer only limited insight into the needs and behavior of top management in today's organizations, let alone the office of the future. We currently lack theories and models with substantial empirical support with which to anticipate and evaluate the impact of office automation on organizational communication systems.

WHAT ARE THE OBJECTIVES OF CURRENT RESEARCH?

The remainder of this paper describes a project under way to anticipate and evaluate the impact of office automation on top management communication in a large decentralized organization. This project is designed to lay the groundwork for management planning concerning the use of information technology to support unstructured management decision making and communication. The primary objective of the project is the development of a behavioral science model within which

- management communication problems can be anticipated,
- solutions can be proposed and compared,
- specific office automation systems can be designed, and the
- impact of alternative systems on organizational effectiveness can be predicted and evaluated.

Such a model provides top management with an alternative to computer vendor systems analysis. This project incorporates several theories of organizational behavior and human performance. It is directly responsive to recent proposals for designing more people-oriented computer systems. Most importantly, the model addresses the complexity of management behavior and of the organizational environment onto which any computer based management support system might be imposed.

This approach to the improvement of managerial communications is based on the contentions that (1) human resources in organizational management are more valuable and less well understood than the hardware and software that might be designed to support them, and (2) that person-computer communication can best be understood and improved by developing a better understanding of how people communicate with each other.

This is an ambitious project. No behavioral science or management science theory has offered a viable solution to the problems we are addressing. Surprisingly few have even tried. Our confidence in success is based on development of a new research methodology and on asking, at the outset, what we believe to be the right questions. For example, the models developed in this project should be able to answer the following questions regarding the need for or evaluation of alternative management communication or decision support systems:

1. How would the nature of management behavior be changed?
   A. How would overall efficiency be improved?
   B. How would overall effectiveness be improved?
   C. How would tasks be redefined by the management?
   D. How would behavioral alternatives be redefined and expanded or limited in number?
2. How would manager interaction be changed?
   A. Which responsibilities and interdependencies would be affected?
   B. Which communication patterns would be affected?
3. How would resource consumption be affected?
   A. How would travel patterns be altered?
   B. How would telecommunication patterns be altered?
   C. How would computer resources be utilized?
   D. How would secretarial resources be utilized?
   E. How would managers' time allocation be affected?
4. What would the impact be on attitudes and morale in the organization?
   A. What aspects of the system would meet with strong resistance?
   B. What aspects of the system would be readily accepted?
   C. In what ways would the system support management growth in communication?
5. How would the important relationships between management support, management behavior, and managerial performance be affected?

The development of models to guide the examination of these and related questions is a necessary first step to rational strategic planning for the office of the future. As Peter Drucker has pointed out,

> *The future manager will find the computer as much a fact of life as children today find the telephone . . . the computer is a tool of liberation if used correctly. Otherwise, you become its servant. It should liberate you from being chained to operations and to your desk and enable you to have time for people and for the outside, where the results are.*[11]

Drucker's optimism is encouraging, but unfortunately not supported by the history of the computer industry.

Improperly anticipated and poorly designed, the office of the future may lead to alienation, job fragmentation, regimentation and the 1984 horror of monitoring of all electronic communications. Properly designed, the office of the future could, instead, increase the manager's control over his or her information space, expanding the rich array of communication channels and formats available for effective organizational management. This project is investigating unstructured managerial communication as a first step in anticipating and evaluating the impact of office automation on organizational communication systems. Four basic goals guide this research:

1. Increased ability to deal with more complex information environments without increasing the number of managers in an organization.
2. Freedom for the manager from his or her desk and office as the central communication and information processing station.
3. Consolidation of management communication, scheduling, and decision making activities and support technology.
4. Increased effectiveness and efficiency in dealing with unstructured management communication tasks.

RESEARCH METHODOLOGY

The methodology used in this research is both empirical and theory-driven. It involves unobtrusive observation and analysis of managerial behavior, on the job, over a period of one or more weeks. From this analysis, a scenario is constructed which highlights key management communication activities. Both structure and content of these communications are analyzed to identify opportunities for increasing managerial effectiveness and efficiency. The critical part of the analysis is the construction of models of the individual manager's task and communication structure. Based on these models, additional scenarios are constructed, showing the impact of alternative management communication support technologies for each manager. This approach is holistic in that the full range of organizational and decision making activities is considered. It uses case study analysis to model the complex reality of individual managers.

This data collection process, referred to as "structured observation," derives from anthropological and sociological research. More recently a research group at the USC/Information Sciences Institute has been using a form of structured observation to study human dialogue as a means of improving man-machine interaction.[12,13]

The most significant application of the structured observation methodology in the area of management work activity was a study conducted by Henry Mintzberg, while a graduate student at M.I.T.[14] That study involved the detailed observation and analysis of work activity of five chief executives over a period of a week each. His results provide important motivation and direction for the development of models of managerial communication. Whereas Mintzberg decided to omit from his analyses all interaction between the manager and his secretary and to classify individuals with whom the manager interacted only as outsiders, superiors or subordinates, this project will analyze communications with respect to specific individuals and tasks.

The theoretical foundation on which data collection and analysis are based in the current project includes a novel conceptualization of managerial activity as communication acts intended to accomplish some specific purpose. These purposes are characterized as tasks which may be in various stages of completion, once initiated. The actual model of tasks, their states, and the effects of individual communications will be formulated out of the analysis of structured observation of individual managers on the job. Thus, the data collection is partially structured prior to its initiation and is augmented by interpretations made during and after the actual observation.

One of the compelling virtues of this methodology is its inclusion of far more of the manager's work environment than a typical laboratory study or questionnaire survey would permit. This approach requires cooperation and a high degree of trust between researcher and manager, but offers significant joint learning opportunities for both. This methodology requires a minimum of conscious effort on the manager's part to generate data while maximizing the opportunity for interpretation of ambiguous behavior by the manager. Perhaps most importantly, the methodology eliminates any need for deception by the researcher and assures the manager of full confidentiality at all stages of data collection and analysis. The most serious drawback of the methodology is the enormous amount of time involved in coding and recoding detailed observation data. This extensive analysis is justified by the insight gained into the managerial process with respect to the particular questions and problems being investigated.

Following preliminary analysis of the structured observation data, models of the manager's communication activity and sample scenarios are presented first to the manager and then to a group of managers in the organization. These scenarios are revised and personalized by the group as part of the generation and evaluation of ways for improving unstructured top management communication in the organization. There is considerable evidence that managers react to idealized scenarios about the future with considerably more insight and enthusiasm than they respond to other needs analysis techniques (such as questionnaires and interviews) or functional specifications of proposed systems.[15]

The collection of scenarios agreed upon as representing information processing activities for a variety of managers are then used to define a set of primitive information processing capabilities and a set of necessary hardware capabilities for making the scenarios possible. These primitives and this functional analysis form the basis for an information system design to support and improve managerial communications.

CONCLUSION

An important advantage of this methodology is that it gets many, if not all, of the organizations' top management involved in the iterative design of their own communications system. When an operational system is finally available they should already be knowledgeable as to its functional capabilities and cognizant of the likely organizational impacts.

The action research approach described above leads to the development of systems custom-tailored to the needs of users, even if the primary hardware and software still come from established computer manufacturers. Several key characteristics of the approach deserve reiteration:

1. The spirit of the project is one of cautious and rational planning for the future by development of a necessary understanding of today's top management communication activities.
2. System design proceeds by interpretation of an holistic model of managerial activities. Scenario evaluation and development brings out the values and perceived needs of the managers themselves.
3. Structured observation of management communication behavior and preliminary design specification are not "technology driven," but reflect characteristics of the client organization.
4. The research methodology has the open intention of gradually educating the managers themselves so that they can contribute knowledgeably and efficiently to any eventual system design and implementation.
5. The project focuses on top organization management on the assumption that their time is most valuable to the organization and thus can most significantly be affected by the quality of unstructured communications support systems.

REFERENCES

1. Nanus, Burt, "The Future-Oriented Corporation," *Business Horizons*, February 1975, pp. 5-12.
2. "The Office of the Future: An In-depth Analysis of How Word-Processing Will Reshape The Corporate Office," *Business Week*, June 30, 1975, pp. 48-84.
3. "Toward the Automated Office," *Datamation*, February 1975, pp. 59-62.
4. "Word Processing and the Computerized Office," *Computers and People*, September 1975, pp. 27-30.
5. Ackoff, Russell, "Management Misinformation Systems," *Management Science*, Vol. 14, No. 4, December 1967, pp. 147-156.
6. Argyris, Chris, "Management Information Systems: The Challenge to Rationality and Emotionality," *Management Science*, Vol. 17, No. 6, 1971, pp. B274-292.
7. Argyris, Chris, "Resistance to Rational Management Systems," *Innovation*, No. 10, 1970, pp. 28-35.

8. Tomeski, Edward A. and Harold Lazarus, *People-Oriented Computer Systems,* New York, Van Nostrand Reinhold, 1975.
9. *Patterns of U.S. Economic Growth,* Washington, D.C., U.S. Department of Labor, 1970, p. 33. (Reproduced in Tomeski and Lazarus, op cit.)
10. Mintzberg, Henry, "The Manager's Job: Folklore and Fact," *Harvard Business Review,* July-August 1975.
11. Drucker, Peter F., quoted in Tomeski and Lazarus, op cit., p. 163.
12. Mann, William C., James A. Moore, James A. Levin and James H. Carlisle, *Observation Methods for Human Dialogue,* ISI/RR-75-33, Marina del Rey, CA: Information Sciences Institute, June 1975 (NTIS #A013242).
13. Carlisle, James H., *Why Human-Computer Interaction Doesn't Work Like Human Dialogue,* working paper, presented at ASIS Annual Meeting, Boston, MA, October 1975.
14. Mintzberg, Henry, *The Nature of Managerial Work,* New York, Harper and Row, 1973.
15. Ackoff, Russell L., *Redesigning the Future—A Systems Approach to Societal Problems,* New York, John Wiley, 1974.

The evolving market for word processing and typesetting systems

by J. CHRISTOPHER BURNS
Arthur D. Little, Inc.
Cambridge, Massachusetts

ABSTRACT

The word processing, text editing and typesetting industries have become an important market for computers and computer software. Industry installations are described and sales are forecast for major systems components over the next five years. An evolution is suggested which will link word processing, in-plant publishing, text editing and business data processing systems over the near future.

More than 100 billion words are set in type each year in the United States—about 10,000 times what an average person could read if he did nothing all year but read. There are two interesting facets to this figure: first, it seems to be rising, not only in absolute terms but in proportion to the population. As nearly as we can tell words set per capita in the United States has risen 16 percent in the last ten years, this in spite of increased television broadcasts and a decreasing percentage of the population attending public and private schools and colleges.

The second facet is that with all this information to exchange we are still choosing to set it in type. Over the past ten years we have seen the development of inexpensive display terminals, high-speed non-impact printers and microprocessors which could bring this information directly into the home, bypassing the centuries-old tradition of typesetting. Yet, except for specialized financial applications, we are not using the new technology. Today nearly 70 percent of what you read has passed through a computer in machine readable form at least once, and yet the product does not differ much from its 14th century Chinese ancestor. We are here talking about typesetting not as a dusty curio but as a market for computers, and the reason for this is fundamental to an understanding of how the market will evolve and the demands it will make on the successful vendor.

Let's look for a moment at typesetting: This is an example of how the news might look if it came over communications lines to a home terminal (Figure 1) and this is how the news looks on a typical newspaper page (Figure 2).

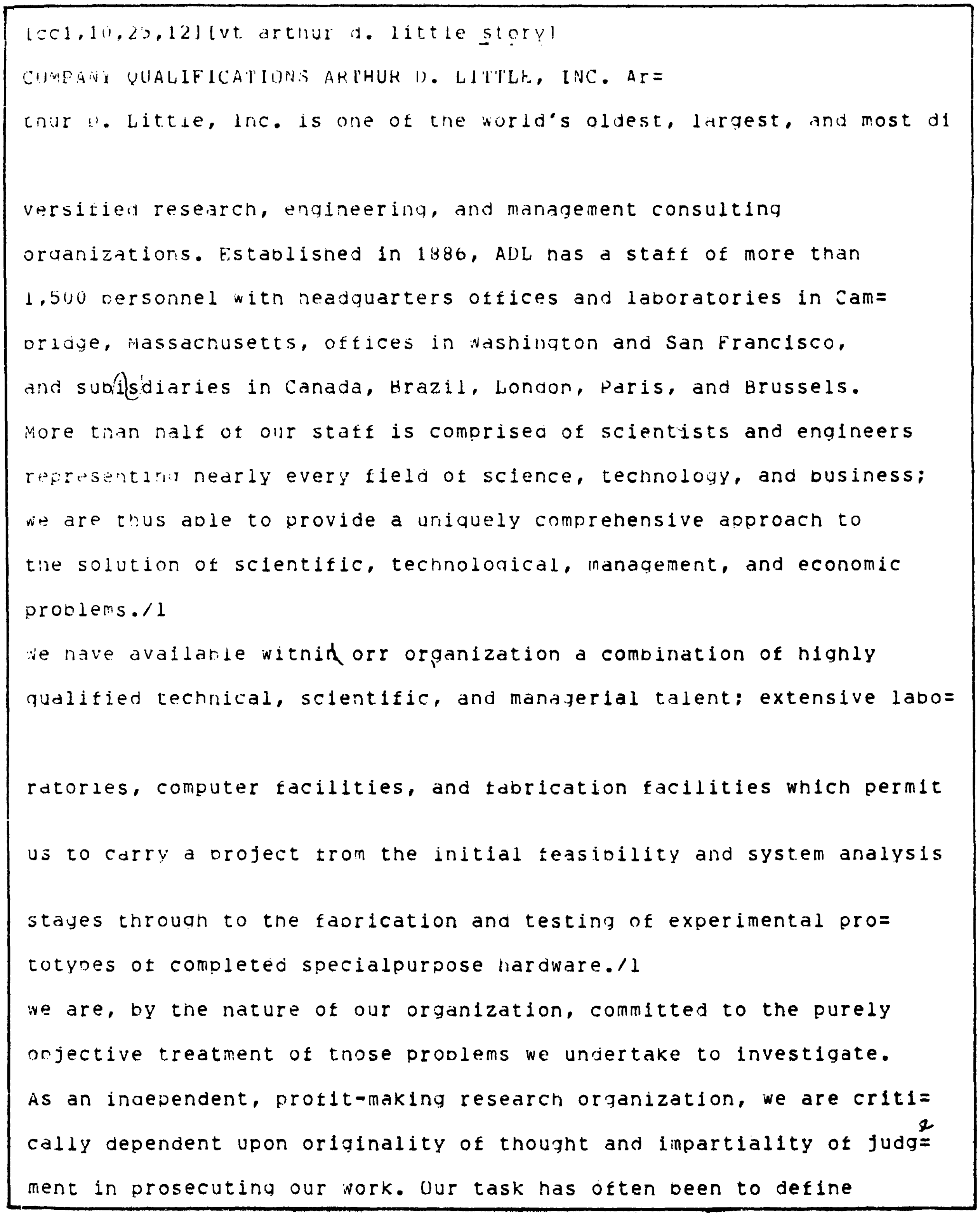
Or this example: An entry from a parts catalog as it might appear on a computer printout (Figure 3) and the same data displayed using typography (Figure 4).

The point, of course, is that typesetting makes it possible to mix sizes, type styles and different layouts in order to present information with greater efficiency, leading the eye, providing ready visual tags to aid retrieval, defining the nature of the information and organizing it for ready reference. Typography is a complex language which can support or overwhelm the message it carries, and mastering this language requires similarly complex commands.

The catalog entry shown, for example, required 104 separate instructions imbedded in the text to select type style, size, line length and so forth, all in composing a block of about 210 words. The development of computer systems to control typesetting has been a surprisingly difficult task. In the ten years since the first work was done on computer assisted typesetting we have only accomplished half the things we knew then to be possible. Hyphenation and justification have been done as well as stored formats, tabular work, run arounds and rudimentary pagination, but there is still no generally accepted system to handle the simultaneous composition of multiple columns, copy fitting, layout assistance, complex chemical and technical typesetting or proofreading, through each was foreseeable as early as 1967.

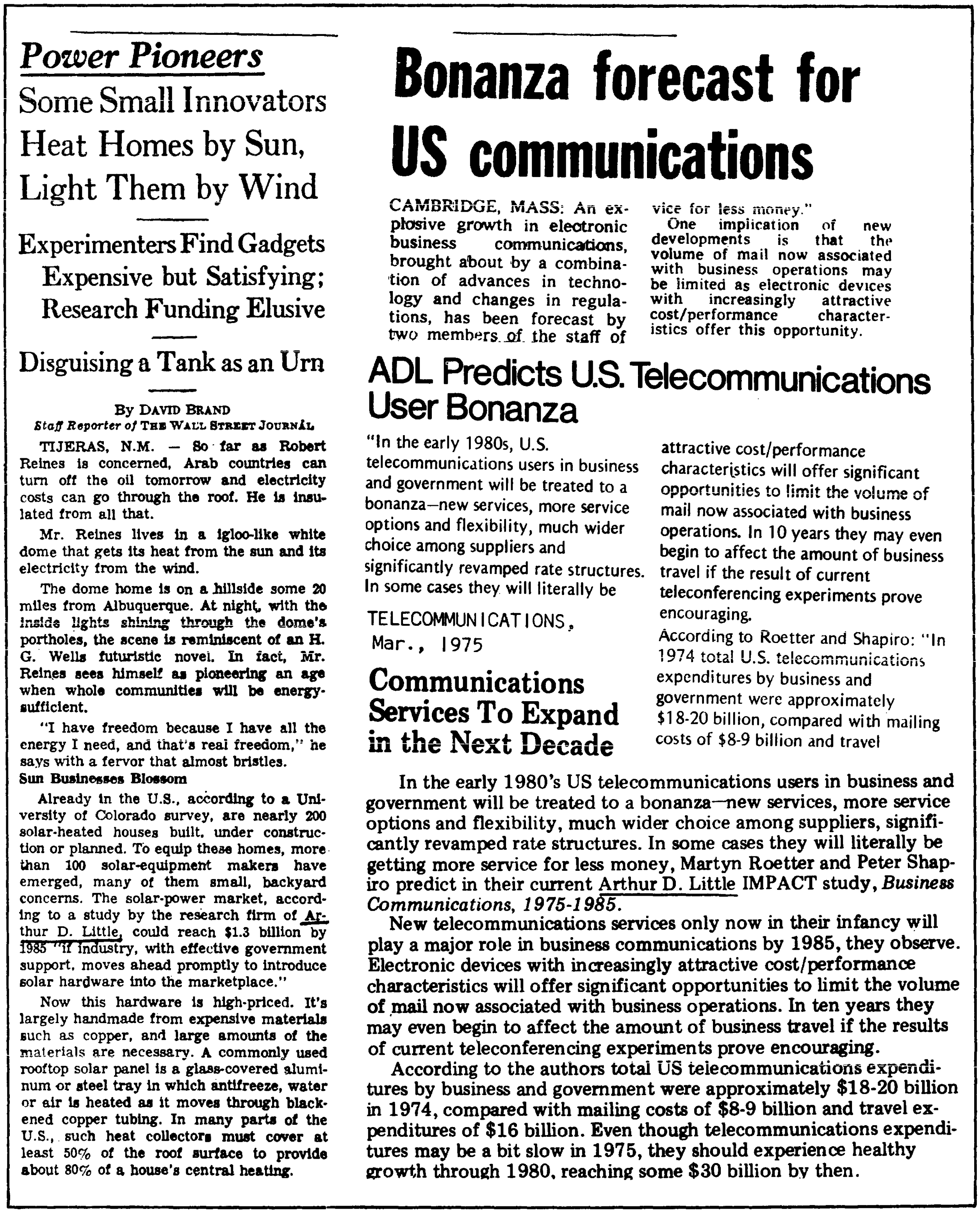
Typesetting systems today are about where business data processing systems were in the days of the 1401, batch oriented, close operator involvement, lots of home-grown software around, little or no full systems integration and occasionally brilliant installations in a general population of ill-fitted, commonplace and troubled efforts.

There are about 800 such typesetting systems installed in the United States today (Figure 5). The earliest of these—the IBM 1130—was equipped with excellent field developed software and became widely popular in the late 1960's although it was only a pro-



```
[cc1,10,25,12][vt arthur d. little story]
COMPANY QUALIFICATIONS ARTHUR D. LITTLE, INC. Ar=
thur D. Little, Inc. is one of the world's oldest, largest, and most di

versified research, engineering, and management consulting
organizations. Established in 1886, ADL has a staff of more than
1,500 personnel with headquarters offices and laboratories in Cam=
bridge, Massachusetts, offices in Washington and San Francisco,
and subisdiaries in Canada, Brazil, London, Paris, and Brussels.
More than half of our staff is comprised of scientists and engineers
representing nearly every field of science, technology, and business;
we are thus able to provide a uniquely comprehensive approach to
the solution of scientific, technological, management, and economic
problems./1
we have available within orr organization a combination of highly
qualified technical, scientific, and managerial talent; extensive labo=

ratories, computer facilities, and fabrication facilities which permit

us to carry a project from the initial feasibility and system analysis

stages through to the fabrication and testing of experimental pro=
totypes of completed specialpurpose hardware./1
we are, by the nature of our organization, committed to the purely
objective treatment of those problems we undertake to investigate.
As an independent, profit-making research organization, we are criti=
                                                                   2
cally dependent upon originality of thought and impartiality of judg=
ment in prosecuting our work. Our task has often been to define
```

Figure 1



Power Pioneers

Some Small Innovators Heat Homes by Sun, Light Them by Wind

Experimenters Find Gadgets Expensive but Satisfying; Research Funding Elusive

Disguising a Tank as an Urn

By DAVID BRAND
Staff Reporter of THE WALL STREET JOURNAL

TIJERAS, N.M. — So far as Robert Reines is concerned, Arab countries can turn off the oil tomorrow and electricity costs can go through the roof. He is insulated from all that.

Mr. Reines lives in a igloo-like white dome that gets its heat from the sun and its electricity from the wind.

The dome home is on a hillside some 20 miles from Albuquerque. At night, with the inside lights shining through the dome's portholes, the scene is reminiscent of an H. G. Wells futuristic novel. In fact, Mr. Reines sees himself as pioneering an age when whole communities will be energy-sufficient.

"I have freedom because I have all the energy I need, and that's real freedom," he says with a fervor that almost bristles.

Sun Businesses Blossom

Already in the U.S., according to a University of Colorado survey, are nearly 200 solar-heated houses built, under construction or planned. To equip these homes, more than 100 solar-equipment makers have emerged, many of them small, backyard concerns. The solar-power market, according to a study by the research firm of Arthur D. Little, could reach $1.3 billion by 1985 "if industry, with effective government support, moves ahead promptly to introduce solar hardware into the marketplace."

Now this hardware is high-priced. It's largely handmade from expensive materials such as copper, and large amounts of the materials are necessary. A commonly used rooftop solar panel is a glass-covered aluminum or steel tray in which antifreeze, water or air is heated as it moves through blackened copper tubing. In many parts of the U.S., such heat collectors must cover at least 50% of the roof surface to provide about 80% of a house's central heating.

Bonanza forecast for US communications

CAMBRIDGE, MASS: An explosive growth in electronic business communications, brought about by a combination of advances in technology and changes in regulations, has been forecast by two members of the staff of

vice for less money."

One implication of new developments is that the volume of mail now associated with business operations may be limited as electronic devices with increasingly attractive cost/performance characteristics offer this opportunity.

ADL Predicts U.S. Telecommunications User Bonanza

"In the early 1980s, U.S. telecommunications users in business and government will be treated to a bonanza—new services, more service options and flexibility, much wider choice among suppliers and significantly revamped rate structures. In some cases they will literally be

TELECOMMUNICATIONS,
Mar., 1975

Communications Services To Expand in the Next Decade

attractive cost/performance characteristics will offer significant opportunities to limit the volume of mail now associated with business operations. In 10 years they may even begin to affect the amount of business travel if the result of current teleconferencing experiments prove encouraging.

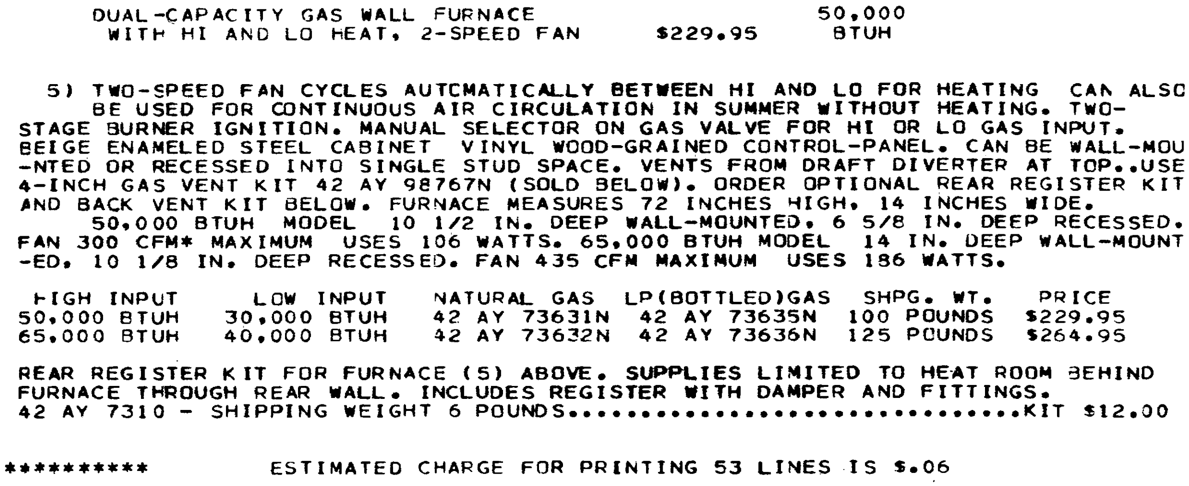
According to Roetter and Shapiro: "In 1974 total U.S. telecommunications expenditures by business and government were approximately $18-20 billion, compared with mailing costs of $8-9 billion and travel

In the early 1980's US telecommunications users in business and government will be treated to a bonanza—new services, more service options and flexibility, much wider choice among suppliers, significantly revamped rate structures. In some cases they will literally be getting more service for less money, Martyn Roetter and Peter Shapiro predict in their current Arthur D. Little IMPACT study, *Business Communications, 1975-1985.*

New telecommunications services only now in their infancy will play a major role in business communications by 1985, they observe. Electronic devices with increasingly attractive cost/performance characteristics will offer significant opportunities to limit the volume of mail now associated with business operations. In ten years they may even begin to affect the amount of business travel if the results of current teleconferencing experiments prove encouraging.

According to the authors total US telecommunications expenditures by business and government were approximately $18-20 billion in 1974, compared with mailing costs of $8-9 billion and travel expenditures of $16 billion. Even though telecommunications expenditures may be a bit slow in 1975, they should experience healthy growth through 1980, reaching some $30 billion by then.

Figure 2



```
      DUAL-CAPACITY GAS WALL FURNACE                    50,000
       WITH HI AND LO HEAT, 2-SPEED FAN       $229.95    BTUH

   5) TWO-SPEED FAN CYCLES AUTCMATICALLY BETWEEN HI AND LO FOR HEATING  CAN ALSC
      BE USED FOR CONTINUOUS AIR CIRCULATION IN SUMMER WITHOUT HEATING. TWO-
STAGE BURNER IGNITION. MANUAL SELECTOR ON GAS VALVE FOR HI OR LO GAS INPUT.
BEIGE ENAMELED STEEL CABINET  VINYL WOOD-GRAINED CONTROL-PANEL. CAN BE WALL-MOU
-NTED OR RECESSED INTO SINGLE STUD SPACE. VENTS FROM DRAFT DIVERTER AT TOP..USE
4-INCH GAS VENT KIT 42 AY 98767N (SOLD BELOW). ORDER OPTIONAL REAR REGISTER KIT
AND BACK VENT KIT BELOW. FURNACE MEASURES 72 INCHES HIGH, 14 INCHES WIDE.
      50,000 BTUH  MODEL  10 1/2 IN. DEEP WALL-MOUNTED, 6 5/8 IN. DEEP RECESSED.
FAN 300 CFM* MAXIMUM  USES 106 WATTS. 65,000 BTUH MODEL  14 IN. DEEP WALL-MOUNT
-ED, 10 1/8 IN. DEEP RECESSED. FAN 435 CFM MAXIMUM  USES 186 WATTS.

 HIGH INPUT     LOW INPUT    NATURAL GAS  LP(BOTTLED)GAS  SHPG. WT.   PRICE
50,000 BTUH   30,000 BTUH   42 AY 73631N  42 AY 73635N  100 POUNDS  $229.95
65,000 BTUH   40,000 BTUH   42 AY 73632N  42 AY 73636N  125 POUNDS  $264.95

REAR REGISTER KIT FOR FURNACE (5) ABOVE. SUPPLIES LIMITED TO HEAT ROOM BEHIND
FURNACE THROUGH REAR WALL. INCLUDES REGISTER WITH DAMPER AND FITTINGS.
42 AY 7310 - SHIPPING WEIGHT 6 POUNDS.............................KIT $12.00

**********          ESTIMATED CHARGE FOR PRINTING 53 LINES IS $.06
```

Figure 3

duction system and allowed no text editing or formatting.

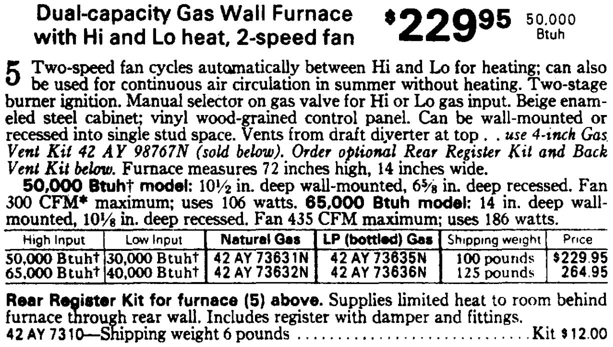
The CSI Photoset and the Digital Equipment Typeset 8 were competitors to the 1130, the Logicon and Sun Com systems were imitators—literally selling the same 1130 with slightly different software. All of these products did hyphenation and justification and formatting, much of which can now be done within the typesetter itself.

Group C on this table represents the main contenders in the market of the early 1970's. Typically they use a 32K PDP-8 or Nova to support a 2.5 megabyte disk, 8 video terminals, an optical scanner and an interface to the phototypesetter. These are dedicated systems with vendor software and almost no general-purpose capability. Rarely are these systems even sold with a compiler. They handle classified ad production for newspapers, text editing, limited file management and occasionally crude operator statistics and production reports.

In Group D are the larger systems in the market today: DEC's Typeset 11, the large Hendrix text editing system, SDC and the large Tal-Star system. Products in this group typically support 32 to 64 video terminals and have been sold almost exclusively to large metropolitan newspapers. Some of them do limited business data processing.

The last group includes the very big systems. DEC has written a full typesetting system for its PDP-10. Harris has built several large systems based on the PDP 11, and there are several major daily newspapers who have written their own software for the IBM 370.

A note on systems which do not appear on this table: IBM has announced but to my knowledge not delivered a typesetting system for the 370 called Printext 370. Univac has announced a typesetting system called Newscom. Dolphin Graphics has acquired the U.S. marketing rights for MOPAS, a European system of some promise and there are at least six other suppliers of typesetting systems in varying stages of specula-



Dual-capacity Gas Wall Furnace with Hi and Lo heat, 2-speed fan **$229.95** 50,000 Btuh

5 Two-speed fan cycles automatically between Hi and Lo for heating; can also be used for continuous air circulation in summer without heating. Two-stage burner ignition. Manual selector on gas valve for Hi or Lo gas input. Beige enameled steel cabinet; vinyl wood-grained control panel. Can be wall-mounted or recessed into single stud space. Vents from draft diverter at top . . *use 4-inch Gas Vent Kit 42 AY 98767N (sold below). Order optional Rear Register Kit and Back Vent Kit below.* Furnace measures 72 inches high, 14 inches wide.

50,000 Btuh† model: 10½ in. deep wall-mounted, 6⅝ in. deep recessed. Fan 300 CFM* maximum; uses 106 watts. **65,000 Btuh model**: 14 in. deep wall-mounted, 10⅛ in. deep recessed. Fan 435 CFM maximum; uses 186 watts.

High Input	Low Input	**Natural Gas**	**LP (bottled) Gas**	Shipping weight	Price
50,000 Btuh†	30,000 Btuh†	42 AY 73631N	42 AY 73635N	100 pounds	$229.95
65,000 Btuh†	40,000 Btuh†	42 AY 73632N	42 AY 73636N	125 pounds	264.95

Rear Register Kit for furnace (5) above. Supplies limited heat to room behind furnace through rear wall. Includes register with damper and fittings.
42 AY 7310—Shipping weight 6 pounds . . . . Kit $12.00

Figure 4

Systems		Systems Installed	Average Price
Group A	IBM 1130	300	$50-75K
Group B	CSI Photo Set, DEC Typeset 8 Logicon-Intercomp, Sun Com	225	$60-150K
Group C	DECset 8000, Dymo CPS 500 & 700, Tal-Star 1000, CSI 24/32, Hendrix, Harris, Atex	180	$100-300K
Group D	DEC Typeset 11, Hendrix 3400, SDC Text II, Tal-Star 4000	60	$300-700K
Group E	DEC Typeset 10, Harris, IBM 370 with field developed software	35	$700K-1.6M
		800	$136.3M

Figure 5—Computer Typesetting Systems Installed 1975

tion, product announcement, and field testing. The most interesting non-system is the Newspaper Systems Development Group, a consortium of eight U.S. newspapers who have contracted with IBM's Federal Systems Division to design and build a full page composition system capable of halftone composition, layout assistance, full page make-up and multi-column composition. It does everything we think a computer can do for a newspaper and a few things over which there is healthy debate. The project is over budget and at least two years behind schedule.

But the U.S. Printing and Publishing Industry has been willing to spend over $700 million on new technology in the past five years (Figure 6), primarily on phototypesetters and small production oriented computer systems. Newspapers have been the most aggressive sector here, and for good reason: payout on a typical newspaper system can be achieved in less than 18 months if the labor situation is right. Newspapers report that conversion from hot metal to computerized photocomposition has cut as much as 50 percent off their composition and production costs, often equivalent to an increase of 10 percent in profit before taxes.

Over the next five years the market is likely to behave somewhat differently. A major increase in small word processing and text editing systems sales is expected, with floppy disk-oriented systems available for below $50,000. Optical scanners are likely to lose sales momentum and may in fact experience declining sales by 1980, functionally replaced by communicating terminals and word processing equipment. And I think it's clear now that Xerox, Redactron, IBM, Digital Equipment and others intend to turn memory and display typewriters into a billion dollar a year business by 1980 with the probable result that at least 25,000 communicating and display typewriters will become terminals for typesetting systems. Phototypesetters will continue to sell, though at a lower price. At the center of this growth is an enormous potential for small and distributed computer systems which not only can meet the composition needs of the user but also can perform file management, message switching, data storage and retrieval by key word or subject as well as inter-computer communications.

It is probably useful at this juncture to point out that there are more than 30 suppliers now trying to capture a share of this market, with only four achieving annual sales over $10 million and at least 12 operating on sales of $2 million or less.

Who will the customer be over the next five years? (Figure 7) Certainly the printing and publishing industry is the primary target and within that the daily newspaper continues to be the most attractive sector, although our own figures are now suggesting a greater saturation of that market than most other forecasts indicate.

COMPOSITION SYSTEMS SALES: 1970-1975

Component	Unit Sales	Value
Typesetting Computer Systems	500	$100M
Optical Scanners	2,000	$60M
Editing Terminals	3,500	$35M
Phototypsetters	20,000	$500M
		$695M

COMPOSITION SYSTEMS SALES: 1976-1980

Typsetting Computer Systems	1,500	$220M
Optical Scanners	2,000	$50M
Editing Terminals (including video display word processing equipment)	25,000	$250M
Phototypesetters	25,000	$400M
		$920M

Figure 6

Market Sector	Number of Establishments	Establishments with 20+ Employees	Annual Composition Volume	Percent of Market
Daily Newspapers	1,770	970	55M Words	86%
Weekly Newspapers	9,500	300	.9M	1
Magazine Publishers	2,500	500	.5	1
Book Publishers	1,200	300	1.9	3
Miscellaneous Publishers	2,000	250	.4	1
Commercial Printing	22,400	3,800	1.0	2
In-Plant Publishing	48,000	5,000	3.8	6
	87,340	11,120	63.5	100%

Figure 7—The U.S. Market for Typesetting Systems

While there are a great number of establishments—an astonishing number, really—less than 13 percent of these shops have more than 20 employees, a minimum number in my opinion to qualify the establishment as a potential site for a computer-based typesetting system. (Smaller shops will certainly buy phototypesetters and some may buy stand alone editing terminals, but few will be able to spend more than $30,000 on composition systems.) We have to ask a second question: How much typesetting is done? Fewer than a thousand newspapers, for example, set 55 million words a year while more than 3,800 commercial printing shops set only a million words. Analyzing the market this way suggests that of establishments with more than 20 employees, 970 daily newspapers set 86 percent of the type and therefore still constitute the giant share of the market. But the same 970 newspapers probably account for 650 of the typesetting systems already installed, leaving an untouched market of little more than 300.

Weekly newspapers, magazines and book publishers present perhaps a more interesting opportunity. There are 1,100 potential systems sites here and we estimate that only 150 systems have been installed. The opportunity for composition savings is not as great in these sectors and the publications are less sensitive to deadlines, therefore less demanding of speed. But major magazines have installed such equipment and are looking now for communications capability, and book publishers are trying to sort out the choice between word processing and communicating text editing.

The In-Plant market at the bottom of this figure is a tantalizing puzzle. Major corporations maintaining their own print shop will certainly buy typesetting systems. How medium-sized businesses ($25 million to $100 million) respond will be influenced dramatically by the shape of new word processing systems. Let me take the last few minutes to describe how I think this market will evolve. (Figure 8)

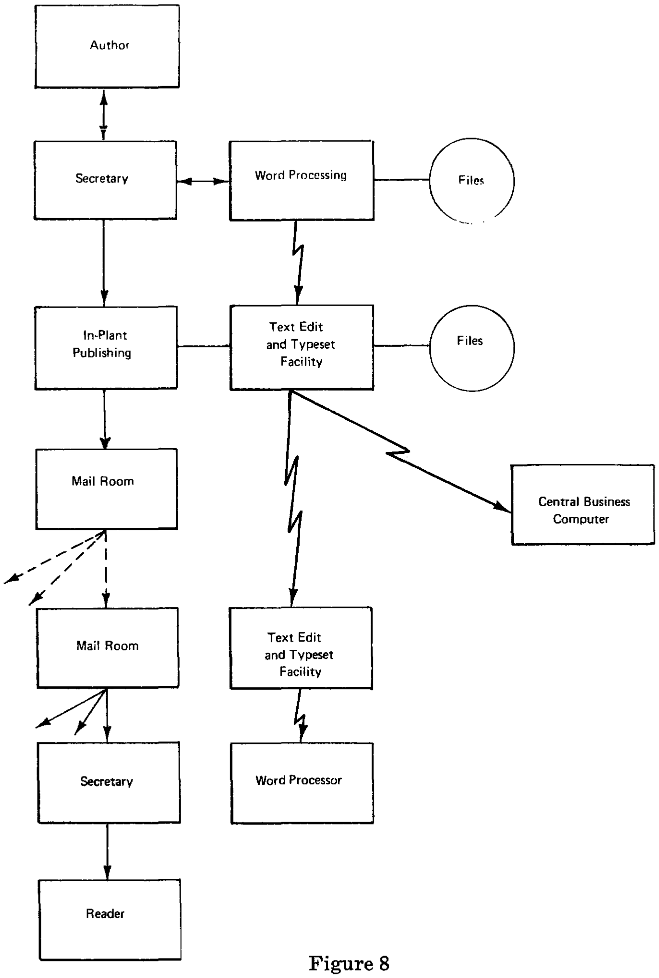


Figure 8

It is probable now that the sale of memory and communicating word processors will grow at a rate of 21 to 25 percent per year, reaching an installed population of 750,000 in 1980. The equipment will be used by secretaries and in administrative service centers to prepare reports, letters, legal documents, telephone directories and memorandums, many of which will be simultaneously stored in machine compatible media like magnetic card, cassette or floppy disk. This is the so-called word processing market.

We expect another development to occur at the same time: the general adoption of mini-computer based text editing and composition systems. By 1980 there will be as many as 1000 of these in medium to large businesses preparing formal reports, pamphlets, manuals, large directories and catalogs. This is what we refer to as the in-plant publishing market. It is obvious, I think, that in a short time these two independent components—the word processing world and the text editing world—will begin communicating. Material drafted on word processors will be composed in a central facility, long contracts stored in a central computer will be retrieved by the word processor to be revised. Product definitions will blur—"Is it an intelligent terminal or a communicating word processor." Alert suppliers will identify and provide a full range of compatible products and we will hear advertising slogans that talk about a "total information system."

And there is another development possible. In our own work evaluating and sometimes designing such systems we have seen cases where communications capability was required not only from the remote terminal to the central facility, but from the central facility to the main business data processing system and occasionally from facility to facility over packet switched communication networks or even dial-up lines.

What we are talking about here is a new business communications system that will provide a rapid, low cost alternative to the present process of typing, duplicating, mailing, distributing and filing. This has some important implications for those who would participate in the market. It means message switching, privileged access, hierarchical organization of files and cross indexing, all this in addition to the communications and support systems.

For years we have used the phrase data processing to mean the manipulation of measurements. With the rapid introduction of computers to typesetting, text editing and word processing we are beginning to process ideas, to gather and select them, store and retrieve them, to format and present them in a way that will enhance their meaning. The associational structure of ideas and messages differs profoundly from the structure of numbers and we will need new strategies for storage and retrieval. The typography of an idea is no less important than the design of a complex statistical report, so we will need new commands, perhaps a new composition language. But the opportunity is enormous and the rewards for both business and society are rich. As computers become smaller, cheaper and more powerful we have the chance to use them for more than the measurement of work accomplished, we have a chance to move them out to the work site itself to speed and clarify the communications on which that work is based.

The computer as a tool in the processing of text for periodical publications

by WILLIAM J. HAMMOND
Publisher's Compu-Type Service, Inc.
New York, New York

ABSTRACT

This paper traces the introduction and initial applications of computer technology into the magazine publishing industry. It reviews developments leading up to our present state-of-the art. It continues with an overview of today's systems. And, it concludes with a preview of future advancements designed to enhance the publishing of periodicals.

Since its introduction into the typesetting process in the early 1960's, the computer has proven itself to be not just a useful tool, but an almost indispensable one, in all areas of the publishing field. The techniques of setting type photographically had been successfully mastered in the late 1950's. The marriage of the computer to phototypesetting freed both the typographer and the publisher from the limitations and the constraints of three dimensional "hot metal" typesetting.

The first functions assigned to computers were those involving "end of line" decisions resulting in the justification of a line of type. Justification is the process of making a line of type fit evenly within the specified line measure. The computer freed the keyboard operator from having to determine when and how to hyphenate words, and from calculating how much space to allow between words. Thus freed, the operator could concentrate on improving the speed with which he could convert a manuscript into coded paper tapes to be processed by the computer. A new, justified paper tape version of the story could then be typeset on a phototypesetting device.

In the late 1960's, automation of typesetting processes began to make real progress as a result of the introduction of minicomputers. The "minis" may not have been as fast at internal computations as their big brothers, but they were fast enough to perform the many repetitive calculations involved in the justification process tirelessly and accurately, and still keep up with the relatively slow speeds of input and output peripherals. The comparatively low cost of minicomputers permitted the widespread introduction of computerized typesetting systems into an industry composed largely of small shops.

Initial acceptances of computerized typesetting in the publishing field were in the newspaper, book and catalog fields. Newspapers were quick to capitalize on the speed advantages offered by computers while books and catalogs found justification in their use based upon reprocessability for republication purposes. Periodical publishing lagged behind in its use of the computer because of its one-time usage of typeset matter and the relatively high degree of alteration of text prior to publication.

The introduction of "minis" not only permitted the widespread use of computers in the industry, but allowed systems to be developed with sizable on-line mass memory and more sophisticated programs to handle storage and editing requirements as well. Let's look at editing for a moment. Editing is the process of making words suitable for public presentation. It does this by performing a number of functions such as: selecting, compiling, writing, rewriting, altering, and formatting. The purpose of editing is to bring about conformity with a set of standards to suit a particular purpose. These standards, generally set by the publisher, might include: clarity of meaning; accuracy of spelling and hyphenation; and an effective or pleasant appearance of the final printed product. Editing is, in fact, a process of manipulation; manipulation of text, of words, of data. And, data manipulation is a process that readily lends itself to the electronic technology of computer processing. It is only natural that computers should prove to be useful and valuable tools for facilitating the text editing process.

Today's computerized typesetting and text processing systems are a far cry from those of 10 years ago. A modern system will include one or more powerful minicomputers. It will accommodate a wide variety of input and output devices. It will employ large random access disc systems for one-line storage and video displays for on-line editing and updating of files. And, it will have data communications capabilities to handle remote input and output to remote printing and publishing locations.

The modern computerized typesetting and text processing system affords periodical publishers and editors a number of facilities to improve their functions.

First, and foremost, is control. For the first time in many instances, control of how, when, and where typesetting is done is in the hands of the periodical publisher. Consistency and accuracy in typographic specifications and style integrity can be guaranteed. A publisher no longer has to depend upon a particular shop, or more likely a specific individual, to interpret an editor's marginal notes concerning style. A few simple control codes can cause the computer to recall preprogrammed rules concerning: the type faces and sizes to be used; the relationship of various elements of the story; rules for the usage of intraword spacing; specific rules for word hyphenation and exceptions to those rules, etc.

The physical layout of text columns on a page is easily controlled. A predetermined size and shape for any column of text can be given to the computer and it will mold the text to fit the desired configuration. Both regular and irregular shapes of blocks of text can be used, resulting in more artistically pleasing relationships between text and illustrations on a page. Entire pages can be composed electronically with all typographic elements assembled in their proper position, thus saving both time and cost of assembling the elements by hand.

The correction processes are greatly facilitated by changing only the words or the individual characters requiring change. Formerly, changing even one letter on a line necessitated resetting the entire line; changing a word often involved resetting several lines. One of the great dilemmas of a publication editor has finally been resolved. Often, an editor had to decide between making the most effective change from the point of view of clarity of meaning, which could result in massive resetting of type, or making a less effective change and avoiding costly and time consuming resetting. Since in today's systems, the text stream and configuration of text columns operate independently, the editor no longer need hesitate to make the most effective changes.

We are living in the midst of what has often been referred to as an "information explosion". The need to know is steadily growing, and timeliness is being equated with newsworthiness. This is as true of the monthly trade magazine as it is of the national news weekly. The modern computer system is satisfying this need too. It has permitted tightening of schedules to the degree that news stories can break within hours of the time the presses must roll and still make the edition. Turn-around times for both original typesetting and corrections are such that many news magazines have justified systems on that basis alone.

Freedom from fear was promised 200 years ago, but it has only recently found its way into periodical publishing. With a modern computer system incorporating a high speed data communications network, publishers of large volume national news weeklies have been able to print identical, accurate versions of the news in multiple printing locations. They can now enjoy the advantages of multiple distribution points without the fear that different printers may have slightly different versions of the news. Control is in the hands of the publication. All magazine publishers, at one time or another, know the fear of having to move from one printer to another. Invariably, the most traumatic part of the move is the disruption and chaos resulting from moving the typesetting operation. "That little old typesetter with the green eye shade is the only person in the *world* who knows what the editors want!" With an in-house, or nearby service bureau handling the typography, the transition can be made without having to change editorial practices or procedures. Possibly the greatest fear of a publisher is the sudden, unexpected loss of services of the printer. Again, with the service bureau at hand, the publication is free to locate and switch to a new printing facility on a moment's notice without fear of major disruptions in its editorial process.

Control, freedom, and flexibility make the computer an invaluable tool in the periodical publishing industry. These factors have taken typography out of the manufacturing process and put it back into the publishing process where it started.

What of the future? What additional benefits can the publisher look to derive from the computer?

Many of the future benefits of computer usage are on the drawing boards and in the laboratories right now. The periodical publisher can look forward to systems that not only generate his typography, but also produce his illustrations; thus giving him the ultimate in control of the pre-press processes. In the area of marketing and distribution, computers will permit the manufacturing of personalized editions of his magazine for each individual subscriber. Advertisers will be able to direct their messages to highly specific audiences getting maximum effective exposure for their advertising dollar. Through modeling techniques, publishers will be able to specify the optimum combination of components in an issue, or they will be able to predetermine which of a number of suppliers is best able to produce a particular product. Of course, data base publishing will come into more and more sophisticated use. Libraries of past issues can be easily retained on magnetic tape with possibilities opening up for indexing, abstracting, and reprinting in new formats. Finally, with the perfecting of graphics generating typesetters, micro-publishing will begin to make inroads into conventional publishing methods. Someday, maybe soon, magazines will be entirely produced on microfilm or transmitted electronically directly into the subscriber's home.

If this sounds like "blue sky," just reflect for a moment on how far electronic technology has progressed in the past ten years, or even in the past five years. As technology continues to make giant steps in progress, that "blue sky" will rapidly come down to the solid reality of ground level.

The integration of microfilm and the computer

by DENNIS R. NEARY and TERRENCE H. COYLE
Eastman Kodak Company
Rochester, New York

and

DON M. AVEDON
National Micrographic Association
Silver Spring, Maryland

ABSTRACT

A discussion of the nature of information and how data flows through a basic data processing system. The key functions of INPUT, PROCESSING, and OUTPUT are reviewed, with emphasis on data capture and output reporting. After a typical information system is described, the various places where micrographics fits are analyzed. Microfilm is shown as a complement to the modern information system. The basic micrographics concepts are discussed. Rotary and planetary cameras are explained, as well as the various microforms that they can be used to create. Coding techniques and retrieval equipment are reviewed, and the value of microfilm as a storage medium is explored. The nature of the COM concept is explored to establish where and in what situations it may, or may not, be valuable. Cost justification is explored as well as systems benefits. COM technology is explained, and the future of COM will be projected. An exploration of the concept of capturing source documents in random sequence on microfilm and using a computer generated index to assist in retrieval. The actual hard wire interface of a computer and a microfilm display device will be discussed. The future of computer assisted retrieval will be projected. The integration of data capture and microfilm will also be discussed. A projection on the status of data processing five years from now, and how microfilm will fit in the information systems of the 1980's.

INFORMATION FLOW ANALYSIS

In analyzing any basic data processing system, the major operations can be divided into input, processing, and output. Data is abstracted and captured from source documents, converted to machine language, and electronically transferred to the central processing unit, which processes that raw data in conformity with a standard application program. As a result of that processing, master files are updated and maintained, transactions are reported, and current status is updated and reported. The reporting has been done historically on paper and more recently with video images on CRT terminals. A third output image that has gained prominence in the last few years is the micro-image, created through computer output microfilmers.

The data processing system has undergone an information explosion. Bulging files, misfiled papers, long searches through millions of manila folders in thousands of offices all over the world, reams and reams of computer output, and expensive, complex on-line systems have strangled the typical modern business. Let's take a closer look at a typical data processing system and see where the micro-image can begin to increase the effective utilization of the entire system.

Information is first abstracted and captured from the source document. Keypunching, key to tape, key to disc, on-line terminals, and optical scanning are the major ways in which information on the source document is converted to machine language and ultimately transmitted to the mainframe. After the information is fed into the computer, what remains is the staggering problem of sorting, filing, and controlling the ocean of source documents. See Figure 1.

Source documents can be categorized as bulk file (unit data) or folder file (co-related data). Generally, bulk file consists of one type of document, and retrieval requires manual search of a paper file for one or more of these documents. Examples include checks, sales slips, time cards, stock certification, and the like. Folder file describes those groups of source documents which contain several types of inter-related data and require merging or updating. Examples include loan application files, insurance policy files, order entry files, and so forth. A key point in the definition of folder file is that folder file information requires inter-active updating. When a folder file becomes a dead record,

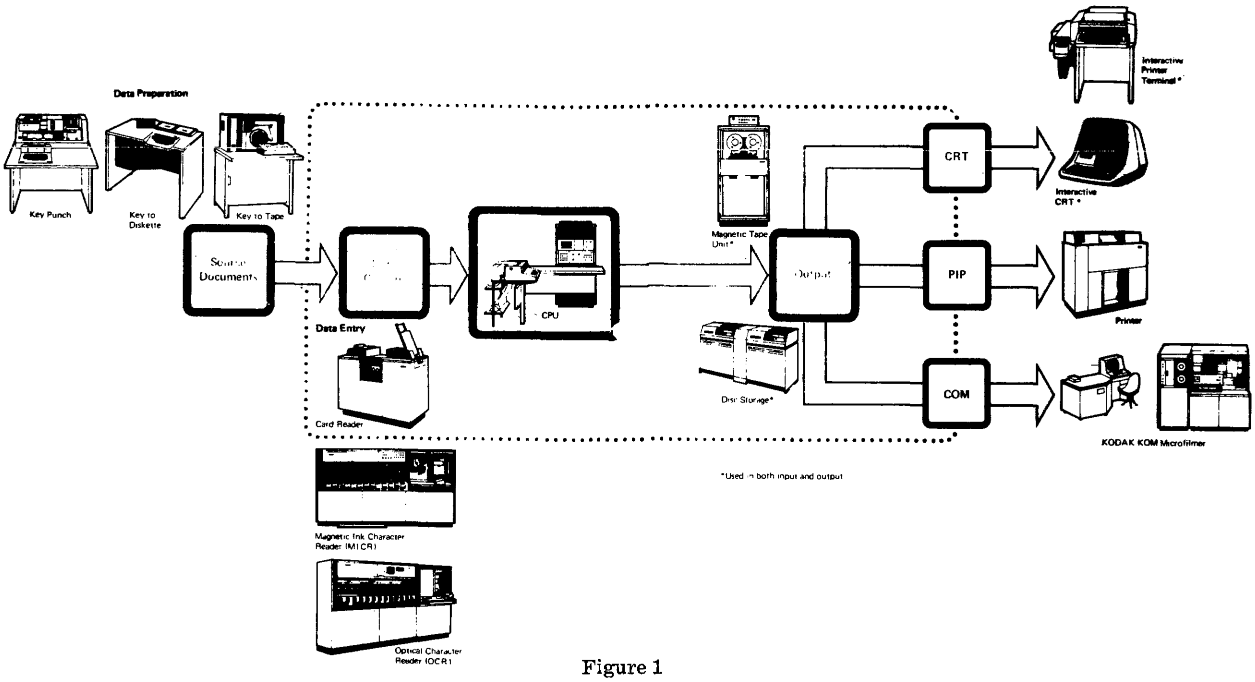


Figure 1

or purged information, it is then considered bulk file information. Regardless of whether a source document is maintained in a bulk or folder file, the micro-image can provide two major benefits: space savings and file management.

Space

The benefits of space savings are important to any growing business.

1. *Opportunity Cost*—The Micro-Image provides the opportunity to use expensive floor space for generating profit, not storing files.
2. *File Location*—The Micro-Image condenses files so that they can be located near the end user.
3. *Duplication*—Too many duplicate copies of dead records are a problem in many office environments. The Micro-Image solves the problem. The user can retain one microfilm copy and create paper copies only when needed. If duplicate copies of the entire file are required for several users, they can be generated at lower cost, occupying less space.
4. *Expansiveness*—Files can become so expansive that they not only occupy valuable space, but become unmanageable with both on- and off-premises storage. The Micro-Image can provide a 98 percent space savings, and virtually eliminate the need to selectively purge paper files.

FILE MANAGEMENT

The benefits of file manageability are benefits affecting the company's lifeblood, the flow of needed information.

1. *Operating Costs*—Paper files are labor intensive, making for a waste of talent in an organization. The Micro-Image with automated filing and retrieval helps reallocate talent, saving money for the prospect.
2. *Accountability*—Files document business operations, providing the basis for answering customer, management, or interdepartmental inquiries. The Micro-Image provides *audit trial control* over files, and eliminates the national average of nearly six percent of misfiles and lost documents in the source document area. The Micro-Image provides file integrity.
3. *Manageability*—The Micro-Image organizes files so they are of manageable size and format. And it pulls these files together so that they can be easily co-related on request. The Micro-Image manages the file through automated information handling.
4. *Security*—Paper can be easily tampered with, stolen, or destroyed. The Micro-Image is discrete, easily and safely stored, and can be duplicated with extra copies kept off-premises.
5. *Convenience*—The Micro-Image assures that the record is there, or it simply does not exist. Records can be accessed faster and more easily with any of the available formats.
6. *Human Factors*—Paper is essential. But after a message or order has been communicated and documented on paper, the *manual* filing process remains. Automated Micro-Image systems can also provide job enrichment to reduce employee turnover.

Microfilming of the source document can take place either prior to or subsequent to data capture. Microfilming can be performed in such a manner that the information is accessible through various techniques which will be described later in this report. However, one approach which is gaining prominence in the recent past is the filming of the source documents in random order, and the assignment of a film address, a roll and frame number. This film address is captured with the other information passed on to the computer which in turn creates an index to the address by any number of parameters which the end user might require. This index can be reported out on paper, or can be maintained on-line, and allows the user to access information from a random film file with the assistance of the computer generated index.

After the information is processed and master files are updated, the computer produces a large number of reports. These reports are typically printed out on an impact printer on paper. Another approach is to maintain the report information on-line in a direct-access storage device and reference the information selectively through a computer terminal. The disadvantage of printing on paper is the tremendous time it takes—approximately 1,000 to 2,000 pages per hour—and the cost of paper and printing, and the problem of maintaining large volumes of paper reports for information retrieval.

The disadvantage of the on-line approach is the tremendous cost to store all the information in the direct-access storage device, transmit the information through a telecommunications system to the terminal, and the software to control the entire teleprocessing system.

A third alternative which has gained prominence in the past few years is a compromise between the two. Rather than maintain the information on-line at extreme cost, it is printed out; but not on expensive paper through a slow printer; but rather on inexpensive microfilm through a high-speed computer output microfilmer. Through this technique, users are finding that they can provide much more information faster to a wider number of users at a much lower cost than they could with either paper output or on-line CRT display.

We have seen where microfilm is used to capture source documents prior to or subsequent to data capture. We also see the part that microfilm can play as an output medium instead of a paper or a CRT image. These two disciplines can also be integrated to provide a total information file which incorporates both paper, electronic image, and a micro-image. Consider source documents which are filmed and the address of that micro-image is passed through the computer to an output report. That output report is produced on a computer output microfilmer, and serves as the index to retrieve the source documents on microfilm. Another group of source documents is captured in the same way, but their particular output report is printed out on hard copy. A third group of source documents are microfilmed, and their address is maintained on-line in a direct-access retrieval system. Now the user can retrieve source document information by either inquiring to a file with an on-line CRT which directs the user to the image on film, or the user can reference a hard copy report which provides direction to retrieval to the source document on film, or the user can reference an index on microfilm created by the COM, which also keys the end user back to the source document on film.

To take this inter-relationship one step further, a CRT terminal can be interfaced to a microfilm retrieval device so that when an inquiry is made to an on-line data base, the resultant micro-image of the source document can be electronically transmitted directly to the microfilm reader and the image subsequently displayed automatically. Systems of this type have been installed and operating for several years, and promise to proliferate in the near future. Now in order to better understand the advantages the micro-image can play in a data processing system, we will take a closer look at how microfilm is actually created and the various ways in which it can be retrieved. We will also look in more detail at the concept of computer output microfilming—how it is done and what advantages it provides over on-line access and paper printing. We will then review the advantages of computer assisted retrieval and project the future of the integration of microfilm and the computer over the next five years.

MICROGRAPHICS—INPUT AND RETRIEVAL

Subject

How microfilm is created and the various ways it can be retrieved.

Topics to be covered:

- Micrographic Concepts
- Rotary and Planetary Microfilmers
- Microforms
- Encoding Techniques
- Retrieval Techniques
- Microfilm as a Storage Medium

Theme

A basic microfilm system follows a parallel path to a basic data processing system. Our first speaker has presented a hypothetical data processing system consisting of input, processing and output. Since my topic follows a parallel path to this flow, I will use this same model to show "How Microfilm Is Created and Retrieved." See Figure 1.

While we are dealing with two distinct disciplines, there is some common ground. You are concerned with the creation, movement and use of "data records"; and we in the microfilm community are concerned with the creation, formatting and use of "film records."

For our purposes today, I will be discussing two types of "film records"—roll microfilm and microfiche. When I refer to roll film I am thinking of a reel of film 16 mm wide and either 100 feet or 215 feet long. And when we discuss microfiche I am thinking of a sheet of film 105 mm high by 148 mm wide. See Figure 4.

The parallel path I mentioned starts when we consider the data for the creation of both the "data record" and the "film record" is common—the same source document. You capture data magnetically and we capture the image photographically on film. The parallel continues until the end of the information path when you visually display "data records" and we visually display "microfilm records."

Let's begin our analogy by looking at input. You must extract pertinent data from source documents and enter it in your information system. You have a number of options—keypunch, key-to-tape, key-to-disk, on-line terminals and scanning. We are also interested in the input document, but from a different vantage point. We are not interested in extracting data, but in capturing the entire document image. To accomplish this task, there is one major option to consider—what type of microfilm camera do I use?

There are two basic categories of source document microfilmers—planetary and rotary. While there are numerous models of each, I will treat them in these two broad categories.

Planetary cameras are manual in operation. Documents to be microfilmed are placed on a copyboard and by depressing an exposure button the image is recorded onto microfilm. This procedure will be repeated from 3,000 to 4,000 times thus resulting in a completed roll of microfilm.

This type of camera is normally used when dealing with large documents or for precision microfilming jobs. Those of you from industrial or engineering firms may be familiar with engineering drawing applications utilizing 35 mm microfilms.

When dealing with normal business documents however, a rotary microfilm camera may be a better alternative. Rotary microfilmers can be used to greatly increase the speed at which microfilming takes place. This is possible because of automatic feeding devices which allow for creating microfilm images on the fly. In this instance, microfilming speeds of up to 615 documents per minute are easily achieved. Rotary microfilmers also offer other advantages, such as packing 15,000 check size images onto a 100-foot roll of microfilm and, at the same time, pictures can be taken of both the front and back of a document. These features, as well as automatic numbering and indexing, can be accomplished with no loss of throughput speed.

Now let's move from the input devices to the medium on which the data will be recorded. In our data processing model, data will be recorded on some magnetic medium—tape, disc, floppy disc, etc. While the medium is generally magnetic, the format does change. The same holds true for microfilm. While the medium is always constant, it can take different formats. See Figure 4.

16 mm roll film is still the most predominant format. Images are organized serially along the length of the film. We will explore how we locate an image in a few moments. However, if serial organization does not suit our needs, the roll of microfilm can be automatically cut up and added to a microfilm file folder called a jacket or film folio. This approach unitizes various pieces of data which have been received over a period of time. In this instance, the film file folder is identified by a descriptor such as account name or account number. The folder then becomes the repository for all documentation relating to that descriptor.

But let's go back to indexing. When you begin processing or moving "data records" you continually address (or index) the current location of that record. When microfilm is the medium, we must also be concerned with where images have been recorded. If a file is already serial we have no difficulty. A simple from-to label on the outside of the reel identifies the contents. The thousands of images on the roll can then be located by techniques such as codeline and odometer indexing. See Figure 4.

Codelines appear as solid bars between the microfilm images. Codelines equate to the parameter by which the file is organized. As the parameter changes to a higher value, the position of the codeline also advances. As film is advanced in a microfilm reader, the codelines appear as a solid bar on the reader screen. By establishing a scale on the side of the reader screen, film can be advanced until the codeline moves into the desired range of numbers.

Odometer indexing operates on much the same principle as a home audio tape deck. By noting the footage locations of where a given song starts, we can at any time go immediately to that piece. The same holds true with microfilm, except that we would be noting where a given range of numbers or names are located. Codeline and odometer indexing are generally used when dealing with general records retention applications. However, as we move into more dynamic environments the ground rules change considerably. In our data processing model, input is normally in a batch or random mode. If we are to microfilm these documents at random, we must then be able to establish film addresses for subsequent retrieval.

Our first speaker established one option for this type of retrieval. When a document is microfilmed it is identified by a "roll and frame number." In the microfilm community this approach is referred to as image control of item address. This film address is carried through your processing cycle and is available by interrogation of your master files, or on a report generated by your system. Once the microfilm address is identified, the proper roll of film is inserted in the microfilm retrieval device and the image control number is entered in the retrieval keyboard. The device then counts forward to display the information located at that address. See Figure 6.

A second alternative for random microfilming operates independent of computer processing for establishing the film location of a document. This technique involves the placement of a binary code on the microfilm adjacent to the image of the document. See Figure 4. The encoding takes place at the microfilm camera as the documents are being filmed. Retrieval is now accomplished by interrogating your film files by the descriptors which were encoded; example, account number, invoice number, etc. At this point we have followed our model system to the point of output. You are all well aware of your options for making information available to your users. Your options include on-line terminals, printed reports, audio-response, etc.

Our first speaker has already exposed you to Computer Output Microfilm (COM) as another option, and our next speaker will elaborate on the benefits of this approach. However, COM also introduces new microfilm formats and indexing techniques which I will cover briefly.

Earlier I discussed two microfilm formats—16 mm roll film and unitized microfilm file folders. Both of these formats are available when utilizing COM. The first roll film is available for serially organized files. The second, the unitized format, is accomplished by automatically inserting images in a microfilm file folder. But now I would like to introduce two formats available from COM—35 mm roll film and microfiche. See Figure 4.

35 mm roll film is similar to 16 mm roll film except that it is approximately two times wider. With 35 mm film, we can now print up to 16,000 images per 100 foot roll which is two and a half times our best alternative with 16 mm film. This is accomplished by printing five "tracks" of film images across the width of the film as compared to the two "tracks" available on 16 mm. The 35 mm five "track" format is particularly desirable where an extremely large data base or report is to be converted to microfilm via COM. Microfiche is a 105 mm by 148 mm sheet of film which can hold up to 288 data pages. Each microfiche is easily titled by report name and from-to report information. An index can also be created to locate the column and row location for each image on that particular fiche.

Microfiche is desirable for a number of reasons:

1. Large reports which will be used by many people are split into more manageable units. A person referencing a microfiche only removes up to 288 data pages from the file. By comparison, a person removing one roll of 35 mm film could be removing up to 16,000 data pages, thereby tying-up the file from any other user.
2. Small reports can be easily produced on one or two microfiche. Using 16 or 35 mm roll film would create many reels of film only partially filled which would have to be maintained in the microfilm file.
3. Perhaps the greatest advantage of microfiche is the low cost viewing devices which are available. Microfiche readers cost from $100 to $300, where a roll film reader could cost $1,000 to $2,000. I will come back to this point momentarily.

The introduction of COM into our model system also provides new opportunities for indexing our microfilm outputs.

Since the data to be recorded on microfilm is a product of your computer system, indexing can be accomplished under software control. Software has been designed by most COM vendors to:

1. Examine data fields to automatically generate code lines which correlate to the parameter by which the file is organized.
2. Examine data fields to automatically generate retrieval code used on Kodak Miracode equipment which corresponds to the descriptors known by the user of the file.
3. Abstract data fields to create an image control index to all data pages on a roll of film. This index page normally appears as the first image on a roll of film.
4. Abstract data fields to create an odometer index to key points in the given report.

An additional indexing technique not mentioned previously is also available when using COM. This technique,

5. examines data fields anywhere on a page and prints it anywhere on the microfilm image as a large eyeball code.

See Figure 4 for a complete review of all microfilming techniques.

The eyeball code is the technique used to generate the title and column heading information on microfiche. However, this technique can be used most effectively on 16 or 35 mm roll film applications to make these formats more retrievable for end users. In this case the large eyeball characters are normally printed at the bottom of a data page and correspond to the information by which the report is organized.

I fully realize that we have examined a great many concepts and new terminology in the past half hour. My intent has not been to confuse you, but to expose you to the tremendous flexibility of the microfilm medium. For you to use this information, it will be necessary for you to delve into the world of micrographics to determine which combination of microfilm cameras, microfilm formats, microfilm encoding techniques and retrieval options will best suit the needs of your organization. I will use an example to clarify this point.

A moment ago I said that one of the greatest advantages of using microfiche is the low cost of the microfiche viewer. However, if we are dealing with an extremely large and centralized microfiche file this may not hold true. In this case it is conceivable that so many microfiche are created that a filing and retrieval problem is created. The result may be that the look-up time to retrieve an image is too long to satisfy a customer telephone inquiry. To satisfy this situation maybe a more automated roll microfilm format will better suit your needs. Indexing and retrieval options such as binary code and image control can be used to support roll film applications. While these options may cost many times more than microfiche viewers, the improved response time may well justify the differential. Only a thorough analysis of all available options will result in the most cost effective format and encoding technique. When we go back and examine our model data processing system in Figure 1, I hope that you would now agree that the "creation and retrieval of microfilm" do follow a parallel path. In many cases the design of effective source document microfilm systems will be dependent on your cooperation and implementation. But hopefully microfilm can do something for you as data processing people as well. This is the subject of our next speaker, so I will turn the program over to Mr. Don Avedon who will be discussing the COM concept and those situations where it may, or may not be valuable.

COMPUTER OUTPUT MICROFILM

We are a computer-dependent society. Thanks to computers, information can be stored, retrieved, manipulated and otherwise dealt with at electronic speeds. We know the information we need is there—and we need it fast!

For computer information to become human readable in permanent form, the output typically takes the form of paper. These massive amounts of computer-produced paper contain more information for use in our society than ever before . . . but in a form that is extremely bulky, cumbersome to retrieve, time-consuming to produce, expensive to store and costly to duplicate. As you know computers operate electronically at the speed of light in manipulating information, but in the printing out of the information, these electronic speeds are shackled to the slow speed of the paper-generating printer. While mechanical printer technology continues to produce higher and higher speed capabilities, accelerated paper printout today falls short of solving the demand for information when it's needed.

Consider the large, familiar accordion-folded computer printout sheets. To get at this information in this form, pages have to be leafed through, heavy volumes must be removed from shelves or file drawers . . . to disseminate information in this form means carrying, mailing, trucking or otherwise transporting bulky, cumbersome material at appreciable effort and expense. To store information in this form means considerable expenditure of space and money.

It does present a problem. We've all become so dependent on the computer that it is now part of our everyday lives both at work and at home. Paychecks, bills, reports, medical records, charge account files, business documentation, scientific data and virtually every type of information we need is now produced by the computer. The net result has been a data bottleneck, instead of a smooth, even flow of information in immediate response to today's high-speed demands. To give us our information, impact printers grind out heavy, bulky, expensive reams of paper that we humans must spend time, bursting, collating, filing, binding, shuffling through and spending lots of money on moving from place to place. Then we have to spend hours looking for what we want in a sea of paperwork. Computers have solved a lot of problems for us . . . but along the way they've created a lot of new ones!

Fortunately, a way has been found to solve the data bottleneck dilemma . . . a way that gives us information when and where it's needed. It is based on the meshing of two of man's most modern technologies—computers and micrographics. We have a three-letter word for it—COM—Computer Output Microfilm. Briefly, COM is the process of converting the output of a computer directly into a human-usable form that can be read by people and recorded on microfilm. Think of COM as THE OTHER WAY to make the conversion. The paper way consists of the computer's familiar accordion-folded page—which may run for hundreds of pages. The COM process reduces the same information to microfilm, achieving an enormous reduction in size. The result is a microform that offers many advantages which we will soon discuss. While speed output is not the major advantage COM provides, it is significant that COM outpaces impact printers and even the newer page printers.

Let's briefly review some speed comparisons and then move on to discuss the many advantages COM provides to its users. The line rate of the COM recorder is 32,000 lines per minute. The typical impact printer in widespread general use in the average Electronic Data Processing installation prints 2,000 lines per minute. As we mentioned, there are page printers, expected to gain increasing prominence which can function at 12,000 to 18,000 lines per minute and beyond. So you see, one of COM's advantages is a speed factor 16 times faster than the impact equipment used in today's typical systems and twice the speed of the new page printers.

So much for speed comparisons. COM's major advantages emerge when we compare COM-produced information with the same information in paper form. We will show how COM breaks the information bottleneck by making information human-usable—cheaper and faster.

Let's consider a typical computer printout of approximately 3,600 pages of 11 by 14-inch paper. The paper alone will cost about $35 and will weigh nearly 40 pounds, filling a large carton. If you want to send it coast-to-coast, you have to spend approximately $11. If you COM-process the same amount of data into microfiche, it will weigh only two ounces and will consist of fourteen 4 by 6-inch microforms which can be mailed FIRST CLASS (and will actually travel as Air Mail) for less than fifty cents. Besides this, the materials cost for the microfiche is under two dollars. COM-produced information in such microforms as cassettes, rolls or cartridges will, of course, reflect reductions in comparison with our carton of paper. See Figure 2.

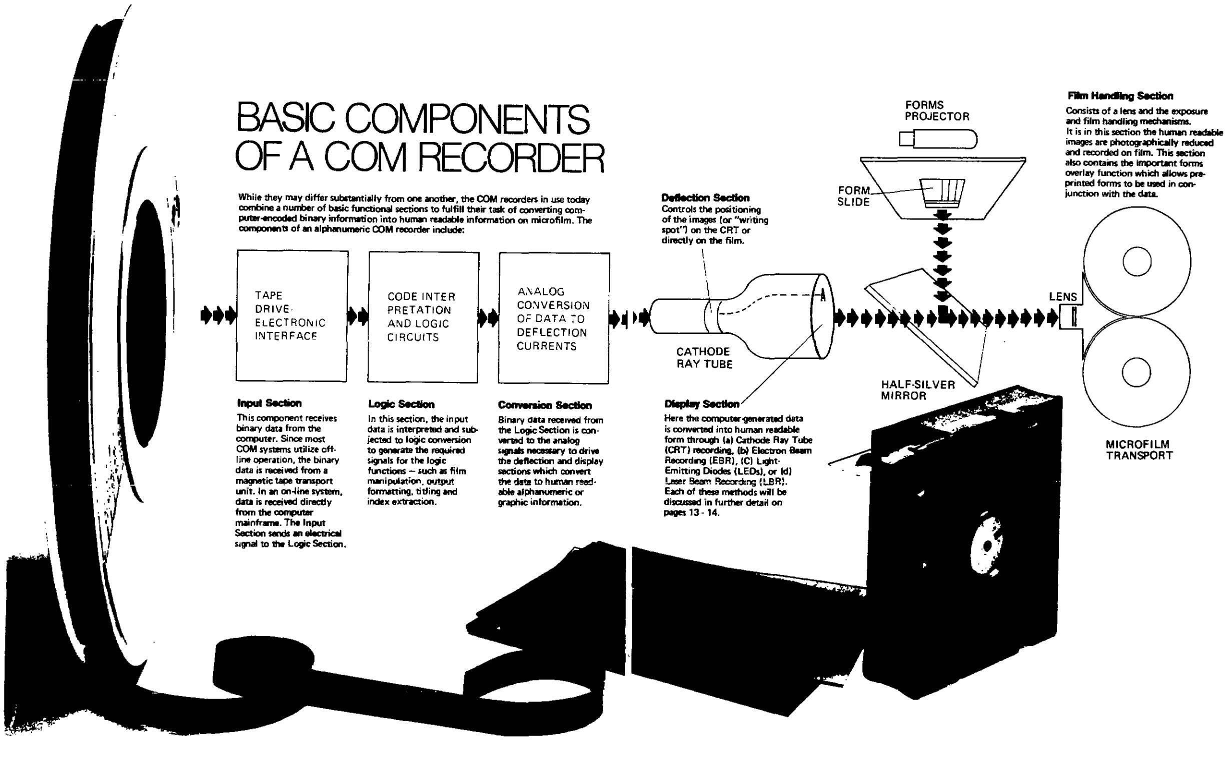
Approx. 3500 Pages (11"x14")	16mm Microfilm in a Cartridge	Microfiche	Computer Paper Printout (preprinted single part forms)
Weight (incl. container or envelope)	One Roll 7 Oz.	14 Fiche 2 Oz.	One Carton 40 Lbs.
Postage	1st Class One Roll 80 Cents	1st Class 14 Fiche 20 Cents	Parcel Post (1000 Miles) $4.25
Materials Cost	One Roll $4.50	14 Fiche $1.82	One Carton $35

Figure 2

We have talked about two advantages of COM: much greater speed and easier distribution. Before we go on to other advantages, let's pause and briefly examine the process by which data from a computer is converted into human-readable form and recorded on microfilm. Although the conversion process can initiate either directly on-line from the computer, it is usually done off-line from a magnetic tape unit for reasons which include not tying up mainframe time and equipment.

From the computer or the magnetic tape unit, the electronic signals that comprise the computer's data move to the COM recorder. In the most widely used method, the signals are reformatted to permit their processing through a cathode-ray tube . . . a CRT . . . where they become human-readable light images which are microfilmed by a camera built into the film handling section. See Figure 3.

The exposed film from the COM recorder now goes through processing. It is usually standard procedure at this point to make a security copy of the information to guard against loss or damage. See Figure 4. Let's look at the aspects of COM processing. For one thing, the system may be set to produce direct positives—that is black-on-white—or reverse forms in which the text or other printed matter appears as white characters against a black surface. Doing this is simply a matter of photographic chemistry. Micrographics specialists maintain that many microforms achieve higher visibility when viewed "in reverse," but a positive black-on-white is better for producing hardcopy.



BASIC COMPONENTS OF A COM RECORDER

While they may differ substantially from one another, the COM recorders in use today combine a number of basic functional sections to fulfill their task of converting computer-encoded binary information into human readable information on microfilm. The components of an alphanumeric COM recorder include:

Input Section
This component receives binary data from the computer. Since most COM systems utilize off-line operation, the binary data is received from a magnetic tape transport unit. In an on-line system, data is received directly from the computer mainframe. The Input Section sends an electrical signal to the Logic Section.

Logic Section
In this section, the input data is interpreted and subjected to logic conversion to generate the required signals for the logic functions – such as film manipulation, output formatting, titling and index extraction.

Conversion Section
Binary data received from the Logic Section is converted to the analog signals necessary to drive the deflection and display sections which convert the data to human readable alphanumeric or graphic information.

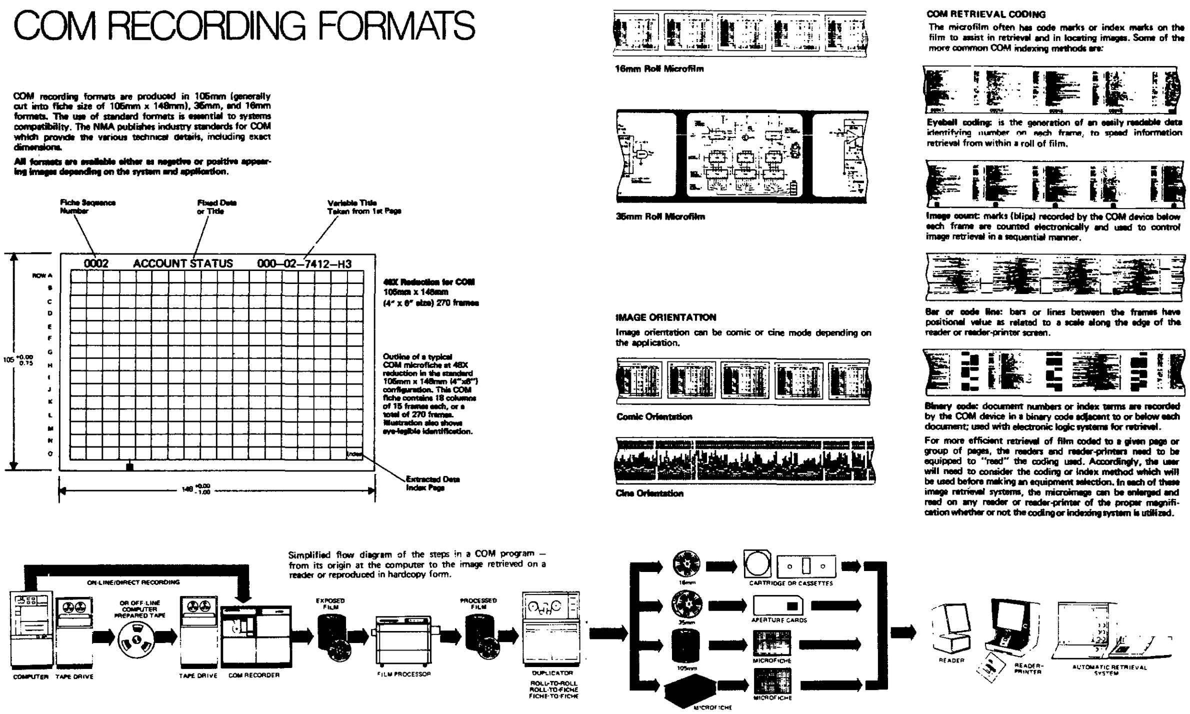
Deflection Section
Controls the positioning of the images (or "writing spot") on the CRT or directly on the film.

Display Section
Here the computer-generated data is converted into human readable form through (a) Cathode Ray Tube (CRT) recording, (b) Electron Beam Recording (EBR), (C) Light-Emitting Diodes (LEDs), or (d) Laser Beam Recording (LBR). Each of these methods will be discussed in further detail on pages 13 - 14.

Film Handling Section
Consists of a lens and the exposure and film handling mechanisms. It is in this section the human readable images are photographically reduced and recorded on film. This section also contains the important forms overlay function which allows pre-printed forms to be used in conjunction with the data.

TAPE DRIVE-ELECTRONIC INTERFACE
CODE INTER PRETATION AND LOGIC CIRCUITS
ANALOG CONVERSION OF DATA TO DEFLECTION CURRENTS
CATHODE RAY TUBE
FORMS PROJECTOR
FORM SLIDE
HALF-SILVER MIRROR
LENS
MICROFILM TRANSPORT

Figure 3



COM RECORDING FORMATS

COM recording formats are produced in 105mm (generally cut into fiche size of 105mm x 148mm), 35mm, and 16mm formats. The use of standard formats is essential to systems compatibility. The NMA publishes industry standards for COM which provide the various technical details, including exact dimensions.

All formats are available either as negative or positive appearing images depending on the system and application.

48X Reduction for COM 105mm x 148mm (4" x 6" size) 270 frames

Outline of a typical COM microfiche at 48X reduction in the standard 105mm x 148mm (4"x6") configuration. This COM fiche contains 18 columns of 15 frames each, or a total of 270 frames. Illustration also shows eye-legible identification.

16mm Roll Microfilm

35mm Roll Microfilm

IMAGE ORIENTATION

Image orientation can be comic or cine mode depending on the application.

Comic Orientation

Cine Orientation

COM RETRIEVAL CODING

The microfilm often has code marks or index marks on the film to assist in retrieval and in locating images. Some of the more common COM indexing methods are:

Eyeball coding: is the generation of an easily readable data identifying number on each frame, to speed information retrieval from within a roll of film.

Image count: marks (blips) recorded by the COM device below each frame are counted electronically and used to control image retrieval in a sequential manner.

Bar or code line: bars or lines between the frames have positional value as related to a scale along the edge of the reader or reader-printer screen.

Binary code: document numbers or index terms are recorded by the COM device in a binary code adjacent to or below each document; used with electronic logic systems for retrieval.

For more efficient retrieval of film coded to a given page or group of pages, the readers and reader-printers need to be equipped to "read" the coding used. Accordingly, the user will need to consider the coding or index method which will be used before making an equipment selection. In each of these image retrieval systems, the microimage can be enlarged and read on any reader or reader-printer of the proper magnification whether or not the coding or indexing system is utilized.

Simplified flow diagram of the steps in a COM program — from its origin at the computer to the image retrieved on a reader or reproduced in hardcopy form.

Figure 4

The next step is duplicating, where one of COM's major benefits becomes visible. With COM, once the master is produced, as many duplicate microforms may be made as needed, from one to hundreds. Instead of activating the computer or off-line tape unit for each set of duplicates required, the task is done by the COM duplicator at a fraction of the cost, and without tying up expensive EDP equipment time.

With COM, the user has a number of options in duplication. The most popular and widely used form typically produced at this stage is the familiar 4 by 6-inch microfiche which originates from 105 mm rolls. The COM user, however, may also select 16 mm rolls for loading into cartridges or cassettes, and another form—35 mm rolls—which are the most frequently used in tab size cards. And so, here you see another feature of COM systems—versatility. Many microforms for many uses! An insurance firm which needs to send data to 1,000 agents would use that many microfiche. A government agency might require a series of cartridges to keep track of personnel records . . . an engineering company can COM-produce its drawings on 35 mm film for mounting on aperture cards . . . and would generate that microform . . . it's all up to the user . . . and COM has the versatility to meet the need.

Now we are at the stage where the COM-produced microforms become the actual working tools for which they were designed. Distribution becomes simple, compared with handling and shipping bulky printed paper. The microforms can be mailed as easily as letters, saving time and expense. In filing, thousands of documents can be kept in a fraction of the space required by paper files . . . and in retrieval, information can be located instantly where it might take hours and sometimes days to hunt through unwieldy volumes of paper.

Now with COM-generated microforms in hand, we can gain access to this information with a microform reader. Selection of a reader is simply a matter of matching the reader to the microform system being used. There are compact, portable readers powered by house current or by self-contained batteries—even some that operate from the cigarette lighter outlet in your car. Most offices make use of desktop readers for personalized use, while many firms maintain freestanding readers for shared use within a group or department—all to insure speedy access to needed information.

And today, there are reader-printers, dual-function units that not only display all of your microform data, but also permit you to make hardcopy printouts when desired. Made at the touch of a button, these printouts may be produced in all standard sizes, including the computer-format size of 11 by 14 inches—all at the same high speed.

Enlarger printers are now in use which print out as many as 2,400 copies an hour from microforms. This speed factor adds high-volume production to their many other advantages and, of course, the over-all COM process has eliminated the need to first print out high volumes of paper or to make multiple printout runs to obtain multiple copies. The ease in retrieving COM-generated information we spoke of earlier can be achieved by many systems now available. There are completely automatic systems which retrieve a microform at the touch of a few keys on a control console. For these automated systems using continuous roll microforms in cartridges or cassettes, COM methods are available to locate the desired images from coding placed directly on the film, an invaluable aid for rapid data retrieval. The user may choose among the various retrieval codes previously discussed.

Let's just pause here and review COM's major benefits.

Speed—Two million lines an hour as compared with 120,000 for impact printing.

Economy—Material costs drop to 20% of paper printouts.

Distribution—Pennies instead of dollars every time you must handle, transport, ship or mail—all with a speed of movement impossible with the former bulk and weight.

Retrieval—Data location in seconds instead of minutes or hours.

Storage—Reduced to less than 1% the space required by paper.

Copy—Hard copies of any microfilmed information on demand in selection of paper sizes, in a matter of seconds.

These advantages have elevated COM into increasingly prominent use in just about every area of business and commerce. Typical COM business applications using aphanumeric equipment include microfilmed parts catalogs, customer/employee/vendor lists, financial records, transportation schedules, accounts receivable/payable, inventories, name/address/account number lists—and general business and governmental documents.

In many applications, the system user requires information to be generated in a special format. When alphanumeric information and forms must be combined COM saves both time and money. With computer impact printers this means that the paper must first be imprinted in order to receive the alphanumeric output. COM eliminates this need by simultaneously generating the column headings, lines, charts, graphs or other special artwork, and blends this with the alphanumeric information in the format that is wanted as the system is receiving the computer output.

Now, COM graphics recorders are available, systems that combine the alphanumeric capability with an unprecedented ability to handle graphics for such applications as scientific, engineering and business needs. This equipment is ideal for visualizing computer output in the form of bar charts, graphs, maps and diagrams. Progressing well beyond their function as high-speed substitutes for impact printers, COM systems are in use that are complete graphic arts recorders. They are able to produce high-quality graphics in any form that can be generated by the computer. COM-oriented graphic arts recorders are now producing engineering plots, line drawings, schematic and wiring diagrams, three-dimensional drawings, color charts, and even pictures, photo composition and movies.

COM applications are virtually limitless. It almost seems that each time a computer specialist and a micrographics specialist meet another COM use develops. Government agencies using COM include the Social Security Administration, which is probably the world's largest user of micrographics, the Internal Revenue Service and many other entities at the Federal, State and Municipal level.

Here, in summary, is the story of COM . . . the story of the solution to a paradox . . . computers capable of providing us with so much needed information can now function free of the data bottleneck that impact-printing imposes. COM systems are now at work serving all of us with the vital information we need in our business and personal activities . . . and the number of systems is growing.

COMPUTER ASSISTED RETRIEVAL

We have seen now the micro-image fits in the data processing system. Microfilm is used to capture source documents and it is used for output reports. The two technologies, data processing and micrographics, can be combined to provide in extremely powerful information storage and retrieval system.

Source documents are filmed in random order and assigned a sequential roll and frame number, a film address. The address is captured along with the data normally abstracted from the document, and processed by the computer. A cross index to the source document by one or more search parameters is created and either stored on-line, or printed out on paper or microfilm thru a COM. See Figure 5. When the index is stored on-line, it can be accessed via a remote terminal which is, in turn, connected to a microfilm display terminal. The user can now request dynamic information from a data base and, thru the index, retrieve source documents to support the data base information automatically.

THE FUTURE

Most people in the data processing community generally agree that we have progressed thru three generations of computers. Some feel that the fourth will be characterized by the creation and effective utilization of the data base concept. The fifth generation will provide hardware and software that will allow the ultimate user to comfortably interact with the data base.

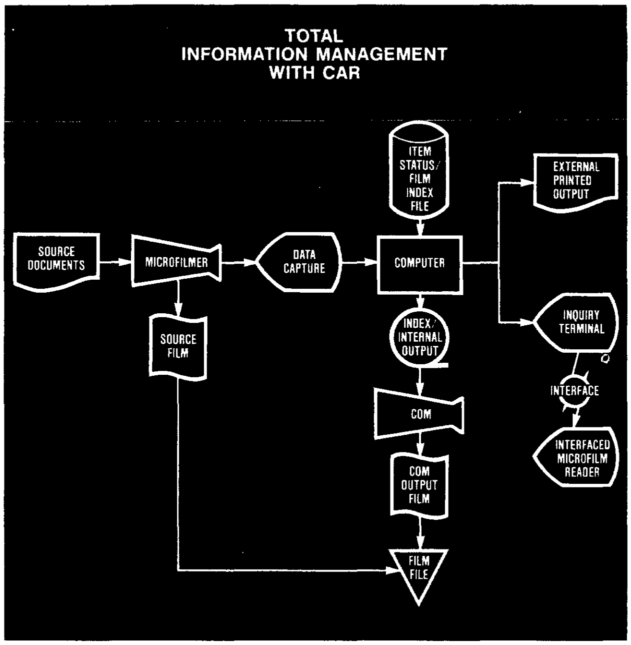


Figure 5

The trend toward large centralized computing systems with extensive data bases appears to be supported by the recent introduction of IBM's 3850 Mass Storage System, an automated tape cartridge library that provides up to 472 billion bytes of memory on line. In order to maximize the effectiveness of these massive systems, efforts will be made in the next five years to distribute processing capability and provide information at decentralized offices for convenience, improved service and reduced operating expense. One of the keys to the success of providing massive computer power to geographically dispersed users lies in the technologically complex area of telecommunications.

Although it is fair to say that problems in telecommunications have been a stumbling block to date, there have been some recent events that seem to project a brighter tomorrow. IBM, Aetna and Comsat have announced a joint venture to provide satellite communications service. Technological advances in the field of Packet Network Switching promise to cut the cost of data transmission in half and several independent networks are already in service. The explosion of microprocessor technology will lead to more intelligent computer terminals and by 1980, many of the inconsistencies that presently impede the progress of telecommunications will be resolved, primarily with the emergence of IBM's SDLC standard. By 1980, we should have a new generation of mainframes which contain millions of bytes of MOS internal storage and billions of bytes of auxiliary disk storage. Although bubble memory and charged-coupled devices also promise to provide vast amounts of inexpensive auxiliary storage, their universal acceptance is still over five years away. On the other hand, the next five years will see significant improvements in MOS and disk storage. We will also continue to strive toward the concept of one massive integrated data base which will be maintained on-line to support a variety of batch tasks as well as interactive inquiry. However, it is doubtful that this utopian approach will be achieved by 1980. Real problems in file protection, contention and logic still remain to be solved. In practice, we will continue to employ, over the next five years, several data bases which are connected primarily by program logic.

There will be a proliferation of a variety of intelligent terminals to provide access to these data bases. Data Collection techniques will also improve as the flexible diskette, optical scanners and electronic funds transfer all mature. There are, however, still some problems. Even though auxiliary storage is decreasing in unit cost, we seem to have a multiple propensity to consume all available storage. After we load in the extensive data bases of the future along with a file of current records, we will still have to find a place to store documents, archival records, and reports of the processed transactions. The micro-image can play an extremely important part in satisfying that need.

The speed of output also continues to retard effective utilization of the computer system. Impact printing is not expected to progress to any great extent. However, non-impact printing should continue to enjoy technological advances that will increase its current capability. Computer output microfilming in particular will adapt to the computer environment and provide more convenient output at reduced cost. This trend in COM will further stimulate the integration of microfilm and the computer.

Most large computer installations and many moderate and small installations have already accepted COM as a viable solution to the expensive cost of paper forms and the bottleneck of impact printing. They are processing transactions, updating the data base, and reporting the transactions on microfilm thru COM. Retrieving the recorded transaction still presents a problem. A potential solution is to use the data base as an indicator of the address of the transactions on microfilm. Thus, the on-line interactive system will use the data base to assist in the retrieval of the micro-image. The computer will also serve to temporarily update the micro-image until it can be re-created.

And what about all those source documents? A discernible trend has already begun. That is to capture the source document on microfilm in random order and assign it a film address, a roll and frame number. This address becomes another element of collected data and is stored in the data base or is processed and incorporated in an output report that serves as an index to the original source document.

The problem today is that it is usually a two step operation, microfilm and data capture. But the philosophy of synergistic design will surely prevail and by 1980 we can expect hybrid, if not integrated devices that simultaneously capture the data for the mainframe and capture the document on a micro-image. Both the data and the document will be forever connected by the film address. Already today, high speed scanners are available with a microfilm camera option.

The discussion thus far, has been oversimplified. There are a multitude of variables and alternatives that will produce many variations of the basic theme. Will the computer be a mainframe, or a minicomputer, or a timeshare service? Will the inquiry be batch processed or interactive? Will the user be local or remote? Will he use a CRT, printer, audio response, or something else? Will the input camera be rotary or planetary, and at what reduction ratio? Will the micro-image medium be roll, microfiche, jackets, aperture cards or something new? All of these alternatives will provide tremendous flexibility in designing information systems to satisfy the needs of tomorrow.

And so what about the question of when to use microfilm versus on-line interactive terminals versus some other alternative? Well, that answer will be dictated by the specifics of each individual application. As we progress, trends will evolve. The integration of microfilm and the computer already has established a viable means to store, retrieve and update source documents in such vertical market applications as law enforcement, order entry, accounts payable and scientific files. Computer assisted retrieval is also currently being used on such computer output files as telephone information, financial statements, inventory control reports and computer generated accounting documents. The number of source document and computer report applications will continue to grow and the scope of their coverage will also expand.

The integration of microfilm and the computer will take time, but it has already begun. The next few years will see several vendors testing the market with hybrid systems that interface computer terminals to microfilm retrieval displays. The terminals will, in turn, be connected to a mainframe, or a mini computer, or a timeshare service. By 1980, integrated devices should evolve that combine the functions of the terminal and microfilm reader. In fact, some are already available today. Data capture microfilmers should also be a reality by 1980, including both the manual and automated high speed variety.

The most significant challenge will be in the areas of system design, software, and file maintenance. Software will be required to generate screen formats to make the terminal easy to use for data collection and retrieval.

Search routines will need to include and/or logic with range capability, as well as the ability to handle update messages. Because of the size of the average file, unique techniques will be required to pack millions of characters into a minimum of storage. As these challenges are met, computer assisted micro-image retrieval systems will flourish.

What the computer and microfilm industries need most now is mutual understanding. They need to realize that they are not competing with one another, and that if they are successful in getting it together, we will all benefit.

The integration of microfilm and the computer will require the cross-training of the EDP and Micrographic communities. This training, in the form of seminars, lectures, formal curriculum and correspondence type courses could be provided by the various EDP and Micrographic vendors. In fact, some commendable work in this area has already been done and has been well received. Now what is needed is a cooperative effort on the part of the various national organizations representing these two disciplines. When workshops on data processing techniques are delivered at microfilm conventions and micrographic seminars are offered at data processing meetings, we will be well on our way to ultimate integration.

The AIDUS system—Automated capture, update and republication of maintenance manuals

by ARNOLD K. GRIFFITH
Information International, Inc.
Culver City, California

ABSTRACT

The AIDUS system is a hardware/software configuration which reads printed aircraft maintenance manuals, provides computer text-editing facilities for their update, and automatically republishes them onto microfilm. Page layout information is entered by a human operator at a digitizer tablet. This information is used to guide an OCR system in the capture of the text. The layout data is merged into the text output file, resulting in a computer readable file containing all the information from the original manuals except for the illustrations. The latter are captured photographically on a roll of 105mm film, which is keyed to the text/page-layout information file by means of bar codes on each frame. Updating of any aspect of the text or layout of the manual may be performed at CRT terminals using text editor programs. Illustrations may be modified by re-drawing, re-photographing, and splicing the resulting frames of film into the 105mm illustration film at the appropriate locations. After revision, the data file and 105mm illustration film are presented to the photocomposition portion of the system, which automatically produces a revised version of the manual, complete with optically merged illustrations and an index accommodating the revised pagination. This system, in use in an actual working environment, allows the update of technical manuals at about one sixth the cost, and with a fraction of the turnaround time, of previously used cut-and-paste methods.

Update and distribution

The motivation for the use of the AIDUS system is the increasing volume of maintenance manuals for out-of-production aircraft in use by the Navy. It is the Navy's responsibility to keep these manuals updated and to distribute updated versions to remote points on land and to ships at sea.

Today fewer new aircraft are developed and only a limited number of each are built. As a result, the Navy keeps them in service for a longer time. This trend to shorter production runs, longer in-service life and the increasing complexity of each new generation of aircraft has placed an ever-expanding burden on the Navy for accurate and timely documentation.

To illustrate the problem of increasing complexity faced by the Navy in publication activity, consider the A-3 aircraft built around 1955. It required 69,000 pages of documentation which was distributed across some 460 manuals. Ten years later the A-7 aircraft required 135,000 pages in over 1,000 manuals. Currently the Navy has approximately 17,000 technical manuals for the maintenance of these out-of-production aircraft, consisting of over one million pages and requiring continual update. This represents a formidable publication task and it is expected to more than double in scope over the next few years.

To facilitate distribution of these manuals, the Naval Air Rework Facility, the unit responsible for aircraft maintenance, implemented a prototype system by which the manuals would be reduced to microfilm and distributed in the form of extremely compact cartridges containing about 2700 pages each. Initially, 22 sets of paper manuals, each comprising 101 volumes, were removed from the A-4 assembly line and replaced by five microfilm reader/printer devices. The paper manuals having been updated and microfilmed, each set of manuals (101 volumes) was now contained on 16 microfilm cartridges, consisting of some 48,000 pages. After a one-year evaluation period, the system was funded and implemented for NAVAIR-wide use and designated MIARS (Maintenance Information Automated Retrieval System).

Having demonstrated the utility of the use of microfilm as an aid to distribution, the Navy turned to the problem of devising a means for the efficient and timely update of the contents of the manuals. After a three year study, it became apparent that this could be effectively accomplished by a total conversion of the text portions of the manuals, with update being accomplished by computer word-processing procedures.

THE ANATOMY OF AN AIRCRAFT MAINTENANCE MANUAL

A major problem in the conversion of the 17,000 maintenance manuals under Navy jurisdiction is the fact that virtually every page of every manual has a unique layout. This is due in part to the intrinsic complexity of the subject matter. In addition, the manuals were created by numerous different manufacturers over a span of more than a dozen years. Finally, the manuals have a variety of purposes ranging from the cataloging of parts to the description of maintenance procedures.

Simple text, constituting less than half of the total page area, is set in any of several dozen type faces and point sizes. This kind of text may be set on a page in two side-by-side columns, or be set in a single half width, one-third width, three-quarter width or full width column. In addition, the text contains figure and table references, whose location must be explicitly captured, since their location affects how the corrsponding figures and tables may be laid out relative to the text.

Approximately one fourth of the total page area of these manuals consists of tables of one sort or another. The most prevalent type of table consists of a running parts list accompanied by a sequence of parts breakdown illustrations. These tables consist of up to a dozen tabulated columns, with a common heading on consecutive pages to identify the contents of each column. Although these tables continue for up to several dozen pages, their handling is complicated by the fact that the contents of the table must remain in step with the accompanying illustrations. In addition, the entries in certain columns of these tables are indented to varying degrees according to which level in a sub-assembly hierarchy the corresponding item belongs. Another type of table consists of a number of text passages or numerical data within a grid of ruled boxes. The structural constraints on such tables are minor. For example the heights of two boxes at different vertical positions within the table need not be the same, and a dividing rule need not extend all the way to either edge of the table. A third type of table, quite common in running text passages, consists of a number of lines of data tabulated in two or three columns.

More than one fourth of the manuals' page area consists of illustrations. These may be halftones or line drawings, or both. They may occupy any position on a page, and either extend the full width of the page or occupy only one column. There may be several different illustrations on a single page with text interspersed.

Besides these major components, the manual pages also contain numerous items such as: running page titles at the top, or bottom, or both; page numbers; chapter titles; section titles; and centered captions over warning and caution notices.

INPUT PREPARATION

The data input process performed by the AIDUS system involves in the reduction of a paper manual to a computer readable data file containing all the text and page layout information, and a roll of 105mm film containing all the illustrations. Although the filming of the illustrations is a relatively simple one-step process, the capture of the text and layout information requires a series of steps. This series of steps may be divided into two phases, the first of which is termed the "Data Preparation" phase.

The first step in the "Data Preparation" phase is the punching of registration holes in the original pages. These holes are punched in a fixed relation to the material printed on the page, and are necessary for controlling the alignment of the text during reading, as well as the alignment of the illustrations during filming. In addition, unique serial numbers are printed on each page. Since each page is processed as a separate unit, three numbers are used as a common identifier for the various files created as the page passes through the successive stages of the data input process. The numbers are used to identify the illustrations as well. When a set of illustrations from a particular page is captured on a frame of 105mm film, it is this number which is bar coded at the edge of the frame.

The next step of the data input phase is the filming of all pages containing text on a 35mm planetary camera. GRAFIX I, the OCR component of the AIDUS system, reads only from microfilm, so that paper documents must be filmed prior to being read. There are, in general, several important advantages to this approach. The most important factor is that pages of any size may be optically reduced to a common film size, making page advance and page finding—the first, crucial stem in the recognition process—fast and accurate. Moreover, measuring the transmitted light passed through the film achieves a more consistent image quality than can be obtained by reflected light from the original paper documents. Another advantage is that the data can be filmed at the originating site and checked for completeness, avoiding the risk of lost or damaged material.

The final step of data preparation is called the "Page Descriptor Entry" procedure, and involves the entering of page layout data by means of a data tablet. Prior to the actual entry, a page is marked up by an editor to record which fonts are present, as well as to resolve certain unusual or ambiguous situations which might slow down the data tablet operator. After markup, the page is aligned on registration pins on the data tablet by means of the holes previously punched in the pages. The data tablet operator enters geometric information by touching various points on the page with the data tablet stylus. Logical information is entered by touching one or another of a "menu" of captioned boxes fixed permanently to the data tablet next to the page. For example, the page number is entered by touching, in order, the various boxes on the menu which contain the digits of the page number. The location of the text is entered by touching a box marked "text" on the menu, and touching the upper left and lower right corners of the actual blocks of text on the page. The layout of the various types of tables is entered by a similar, but more elaborate process, as well as the locations of the tab stops, titles, locations of cautionary notes in the text, various fonts, and other features of the page. The entire Page Descriptor Entry process, exclusive of markup, requires on the average of less than a minute per page.

TEXT CONVERSION

The second phase of data capture consists in reading the text of the manual pages to produce an output file containing text and page layout information, and subjecting this file to a proof and edit cycle to insure its accuracy and completeness.

The reading of the text is carried out on the GRAFIX I,[1,2] a commercial optical character recognition system. GRAFIX I consists of three components. The first of these is a high precision flying spot scanner. The second component is a medium-scale general purpose time-shared computer. The third component is a slave processor called the Binary Image Processor,[4] which performs a wide range of inner-loop recognition and image processing tasks at very high speeds. GRAFIX I has been successfully employed for numerous other multi-font character recognition tasks, and even reads intermixed alphanumeric handprint.[3]

Recognition of the individual characters of the text is performed by a mask-matching algorithm. Recognition is dependent on the successful location of the successive characters in a line of text. This is performed automatically, sometimes by rather complex character separation algorithms, once the appropriate location and angle of the baselines of the entire line of text has been established. This baseline is, in turn, found automatically, given the location and orientation of the entire paragraph or column of text. This latter information is supplied to the computer during the Page Descriptor Entry procedure. The output file contains the passages of text together with imbedded typographical commands, which incorporate all of the page layout information entered during the Page Descriptor Entry procedure. In general, the structure and form of the page layout information, as entered during the Page Descriptor Entry procedure, is quite different from the corresponding layout commands as they appear in the text. But the process of generating these latter commands is rather straightforward. Certain additional commands are put into the output stream based on factors determined during the character recognition process itself. One such is the command indicating the beginning of a paragraph, which is generated by detecting the level of indent, and from certain textual clues.

Besides the text and layout commands, the output file from OCR contains the images of every character, which the system did not recognize, imbedded in the text at the point at which the character occurs.

The second step in the data capture phase is to subject this output file to a "Reject Conversion" process whereby a new file is created identical to the previous one except that the appropriate ASCII codes replace the images of the unrecognizable characters. This procedure is performed by reading in the successive lines of text until one is found with a "rejected" character, displaying this line of text together with the image of the character at a CRT display terminal, and having the operator key in the correct identity of the character at a keyboard. By this process more than 3000 rejected characters may be identified per hour.

The third step attempts to automatically identify recognition errors in the text output file. In certain instances, a positive identification of an error may be made, such as in the case of a numeral "1" appearing in the middle of a lower case word, which almost certainly is actually a lower case "L". These errors are automatically corrected. Another procedure, using a dictionary, flags seemingly unusual words as possibly containing letter substitutions.

The entire file, complete with embedded layout commands, is now printed out on a high speed lineprinter for proofreading against the original. When the dictionary procedure is used, words flagged by the dictionary process are underlined in the listing, and proofreading is largely a matter of examining these underlined words and checking for completeness. The layout commands are also subject to proofreading at this point, since the layout command language was specifically designed with this in mind. Proofreading is followed by an editing of the file at a CRT/keyboard station to incorporate any necessary modifications.

REPUBLICATION ONTO MICROFILM

At this point a manual is transferred to magnetic tape, which is mounted, together with the appropriate 105mm illustration film, onto the COMP80. The manual, including all text, tables, and illustrations, is automatically republished onto microfilm in a recomposed form, complete with illustrations and an index revised to accommodate the new pagination.

Simultaneously, a data tape is created which is virtually identical to the input tape except that page and line breaks are in the same places as in the version being written onto microfilm. This tape is transferred to the GRAFIX I system where it is converted to a slightly different tape form for archival purposes. This archival tape is stored as the digital version of the manual; and it is retained and retrieved at a later date when a revision of the manual is necessary.

The process of creating a file for input to the COMP80 includes the checking of all composition and format statements for validity and plausibility of their respective numerical arguments. In the event that a statement does not pass these tests, an error message is printed, the file in question is edited, and the process is repeated. Since each page of text is converted as a separate data file, the various files constituting a manual are appended in the course of creating this output tape. In addition, since in general several manuals are composed on a single film, the various constituent manuals are concatenated onto a single tape. The selection of manuals to be placed on a single film is based on factors of convenience, and on the restriction that the total number of pages per film must not exceed 2700.

Since output is composed and typeset automatically, hand stripping and pasteup are eliminated. The labor and time normally associated with book make-up operations is drastically reduced. Rather than a six to 18 month production cycle, manuals can be updated and distributed in as little as 60 days.

Another aspect of GRAFIX I/AIDUS output capability is that publication formats can now be standardized uniformly or modified for new distribution requirements. The updated Navy technical manuals will now have a uniform format as documents from a wide variety of sources are recomposed and republished. In addition, a proportionally spaced type font was designed for the microfilmed manuals which attempts to be compact but maximally readable with the MIARS film reader.

UPDATE FOR PUBLICATION OF REVISED VERSIONS

When it is necessary to revise a manual, its archival tape is read back into GRAFIX I disk storage and divided into its individual pages. These pages are edited, and the publication cycle described in the preceding section is repeated. In addition, any revised illustrations are re-filmed and spliced into the 105mm illustration film.

Editing is performed on a per-page basis by GRAFIX I's special editing software. Using on-line CRT/keyboard terminals, editors may work from either the terminal display, hardcopy printout, or composed film proofs of the converted text. Material to be updated can be accessed by manual, section, chapter, and page number. Updated portions on film may be identified by vertical change bars in the margins; previous change bars are automatically deleted in update runs.

Updating functions include text deletion, replacement or insertion; arbitrary layout changes, accomplished by modifying the page layout commands imbedded in the text; modification of tables including content and layout; illustration additions, deletions, replacements or size changes.

New or updated illustrations are filmed and merged with the illustration file previously filmed during Input Preparation. When a particular illustration is supplanted, its successor is assigned a number higher than any so far on the film, and this number is bar coded on the edge. The frame is spliced into the film in place of its predecessor. A simple change of reference number in the text file completes the process. New illustrations may be added by a similar procedure.

CONCLUSIONS

We have described the AIDUS system, which consists of an OCR device, CRT/keyboard facilities for performing text editing, and a photocomposition system. We have shown how, with a judicious combination of filming, human interaction, and optical character recognition, this system is capable of reducing the entire contents of complex technical manuals, including layout data, to an illustration merge film and a computer-processable data file. In this form, the manual may be easily revised and subsequently republished onto microfilm. We have described an actual working environment in which this system is used, namely the update of maintenance manuals for out-of-production Naval aircraft. In this environment, the system performs update and republication at a projected long-range cost of 15 percent of that of cut and paste methods previously employed and at a fraction of the turn-around time.[2] This system appears to have a wide range of application such as parts catalogs, and maintenance and technical manuals of all sorts.

ACKNOWLEDGMENTS

Designers of the system include, besides the present author: Richard Martin, Russell Ham, Ron Traver, David Meredith, Carol Gillespie, and Dan Forsyth. Other contributions were made by: Steve Gray, Mary Strobel, Glen Williams, Nada Berger, and Norm Gilbreath.

REFERENCES

1. Griffith, A. K., "The GRAFIX I Image Processing System," *AFIPS Proc 1974 National Computer Conference*, pp. 267-272.
2. Information International, *The GRAFIX I Automated Data Base Conversion System*, Document 90475-5M, Information International Inc., Culver City, California, 1975.
3. Griffith, A. K., "Handprint Recognition on the GRAFIX I: A Commercial Application of Heuristic Programming," *Proceedings of the ACM Annual Conference*, pp. 368-372, November 1974.
4. Gray, S. B., *The Binary Image Processor and Its Applications*, Document 90365, Information International Inc., Culver City, California, 1971.

The CMU RT-CAD system—An innovative approach to computer aided design*

by DANIEL P. SIEWIOREK and MARIO R. BARBACCI
Carnegie-Mellon University
Pittsburgh, Pennsylvania

INTRODUCTION AND MOTIVATION

As technology has evolved the primitive components available to a digital system designer have changed dramatically. Twenty-five years ago the designer constructed systems out of circuit level components such as resistors and diodes. Subsequently switching circuit level components, as represented by gates and flip-flops, became available as small scale integration (SSI) components. With the introduction of medium scale integration (MSI) register transfer level components appeared: arithmetic and logic units, registers, shift registers, etc. The advent of large scale integration (LSI) has made memories and even processors primitive components from which systems are designed. Two trends can be observed from this technological evolution: (1) primitive components continue to increase in complexity and (2) the rate of introduction of new components continues to increase.

In response to the first trend, designers have been limiting their excursions into switching circuit level design to only small portions of the system (e.g., bus controllers, etc.). In some register transfer level module sets,[1,2] these excursions have been completely eliminated.

Because of the second trend, rapid technology evolution, there is a need to shorten the delay time between the introduction of a technology and its effective use in new computing systems. The design process must, therefore, be accelerated if the potentiality of the improving technology is to be realized.

This paper describes a set of design programs, the RT-CAD system, developed at Carnegie-Mellon. The ultimate goal is to minimize the effect of changing technology by building a Computer Aided Design System that implements a technology-relative design process.

OVERVIEW OF THE AUTOMATIC DESIGN PROCESS

Given the complexity of a digital system, designers have sought to develop automatic means to reduce the cost and time of the design process. The objective was to relieve engineers of repetitive, time consuming tasks such as:

(1) The generation of detailed design information (gate and chip types, etc.)
(2) The control of changes in the design documents
(3) The checking of the system for electrical, logical, and physical compatibility (fan-out limits, etc.)
(4) The generation of detailed manufacturing information (chip placement, board layout, etc.)

This early view of design automation limited itself to filling the gap between the low-level design specifications and the manufacturing data. Behavioral specifications were in the form of Boolean equations and the design programs translated them into their equivalent logic diagrams and wiring lists. Most of the synthesis algorithms at this level dealt with the problem of reducing or simplifying the Boolean equations.[3]

Subsequent efforts were directed towards a system capable of accepting a higher level of behavioral description, although still oriented towards a gate level implementation.[4,5]

Current design automation effort is shifting from implementation in terms of the switching circuit level to implementation in terms of the Register Transfer Level (Although the components have changed, the specification of fabrication information such as PC board layout, chip placement, wiring and cabling, still has to be performed in more or less the same fashion as before). Register Transfer level simulators have preceded this trend by several years.[4,6,7] The closeness of

* This work was supported by the National Science Foundation under Grant GJ-32758X.

RT level descriptions to conventional programming accounts for this early success. Register Transfer level descriptions are easy to transliterate into executable programs in a conventional programming language (e.g., FORTRAN, Algol, etc.), thus providing inexpensive and fast simulation (although in many cases RT languages are compiled directly). Register Transfer level synthesis algorithms have been less successful. A few programs have been developed that take an RT level description as input and compile it directly into a known set of RT level hardware modules (for instance, CHARTWARE[8] and AHPL[9]). Figure 1 depicts a typical RT design automation system. The RT level description serves as input to several software modules. Syntax checking insures a well formed description. Static checking attempts to locate logical design errors (such as deadlocks, redundancy, etc.). The simulator is used to debug the design dynamically. Finally, the description is cast into hardware via the physical design programs.

The essential feature lacking in conventional RT Design Automation (DA) systems, and DA systems in general, is the exploitation of alternative implementations derived from the initial behavioral specification. Consider the augmented DA system depicted in Figure 2. The inputs are the RT level description and designer given constraints. The output is the specification/simulation of the hardware that attempts to optimize the system according to the design constraints. By allowing the description of alternative module sets the system can perform design relative to technology thus speeding up the incorporation of new technology into the design process. Also, such a system will allow experimentation with multiple module sets, each tailored to a specific class of problems. The system would also facilitate the design of the module sets themselves. Since the system operates on a symbolic description of the modules, a non-existing module set can be fed to the system for experimentation purposes. Such experiments will point out the advantages and disadvantages of a proposed module set.

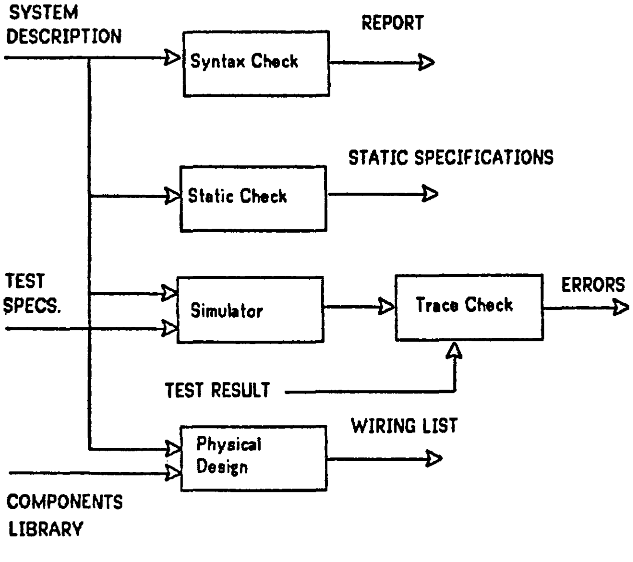


Figure 1—A conventional register transfer level design automation system

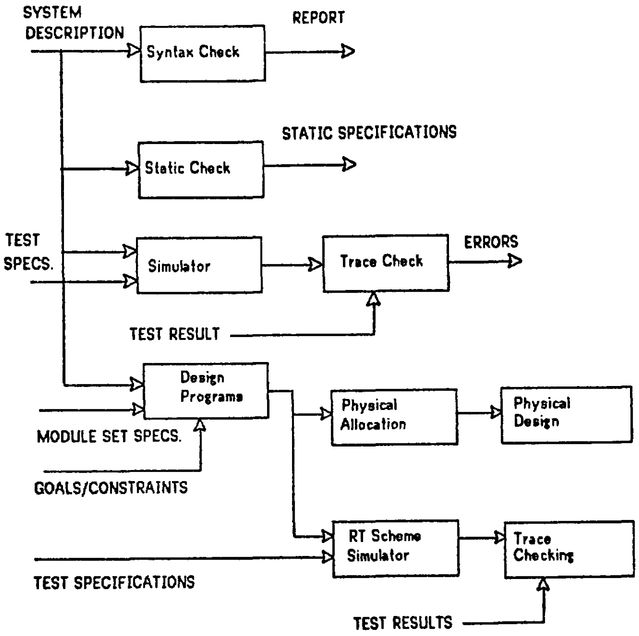


Figure 2—An augmented register transfer level design automation system

At this point it would be instructive to describe the order in which the DA programs are typically used in the design process. This will serve to place subsequent discussions in perspective. Given a computational task, there are usually several algorithms that can be employed. The algorithm that is selected by the designer is described to the design automation system (Figure 3) and placed in a data base. Subsequently all design automation programs will use this data base. A high level simulator can execute from the data base to facilitate user debugging of the initial description.

Next some evaluation and reshaping of the algorithm is undertaken. Analysis tools have been developed to check the algorithm for well-formness (e.g., deadlock conditions, etc.).[10] Perturbations of the basic algorithm can also be attempted such as: series-to-parallel transformations, replacing loop counters by wired-in control, and using table look-up in lieu of computing the value of functions. Thus attempts are made to first

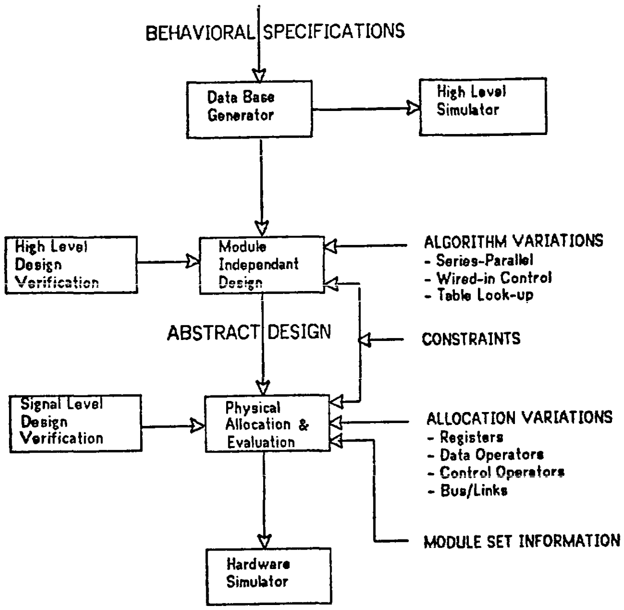


Figure 3—The design process in a RT level design system

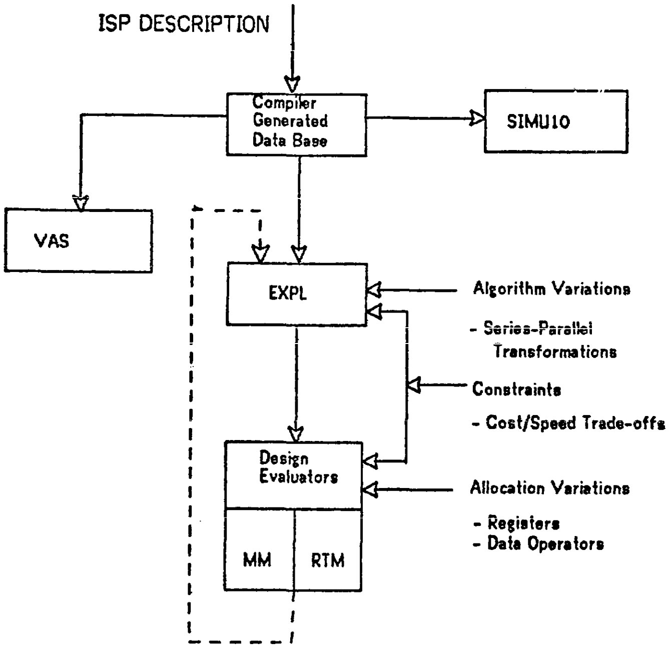


Figure 4—The design process in the CMU-RT-CAD system

bind those design decisions with global implications. While these perturbations can be performed independent of the physical design, the evaluation of their ultimate desirability may depend upon the module set used to implement the final, physical design.

Finally, the actual physical design is performed in terms of RT level modules. The module set can be selected from a library of module sets or a user described set. At this level several forms of allocation variations are encountered:

—Registers. Determine the allocation of the abstract variables to registers and memory.

—Data operators. Determine the number of operators of each type in the design.

—Control. Select control schema from among unary state encoding, binary state encoding, microprogram control, etc.

—Bus-Link clustering. Many RT designs start with a set of registers for variables and interconnect them with links to operators (add, shift, multiply, etc.). After a point the interconnections between certain registers and operators become numerous enough to warrant replacement by a bus.

—Operator interconnection. The interconnection of operators has been shown to have a significant effect on the test generation effort required for the physical implementation.[11]

The signal level design verifier can be used to analyze the intermodule signal relationships in proposed module sets. Even well-established module sets have exhibited deadlock behavior in what appear to be straightforward interconnections.[10]

A first version of the above system has been implemented at Carnegie-Mellon and is shown in Figure 4. The behavioral specifications of the system to be designed are provided in terms of the ISP language.[12] The compiler produces an "object"* program which is then loaded into the data base and manipulated by the different design programs.

The next six sections will treat the applications programs in detail. The third section describes EXPL, a module independent design program that examines series-parallel variations in the original algorithm. The fourth section presents the physical allocators for two existing RT module sets—RTMs and Macromodules (MM). The fifth section discusses the heuristics used by EXPL to explore the design space. Sample design spaces, examples of the application of the heuristics, and some observations are presented in the sixth section. Design verification is discussed in the seventh section. The last section concludes the presentation of the existing system by briefly outlining the remaining application programs.

AUTOMATIC DESIGN SPACE EXPLORATION

EXPL[12] takes as input the object code produced by the ISP compiler, together with a set of user given

* The compiler produces an "object" program in terms of a set of Register Transfer level primitive operations. This program appears in the form of an executable BLISS program where each Register Transfer operation is represented by a call to a user-provided subroutine. By changing the set of subroutines, the compiler can support many diverse activities. The creation of the data base is, in fact, done by a specific set of subroutines. The compiler and the language are therefore independent of the applications. The uniform compiler output and the flexibility of the subroutine-call mechanism has simplified the interfacing to other application programs.

speed/cost constraints/tradeoffs. The compiler output is used to generate a graph representation of the behavior of the system. Subsequently, various series-to-parallel and parallel-to-series transforms on the graph are attempted to establish a new design. Several alternative designs are generated and passed to module set evaluators which complete and evaluate the design in terms of its hardware module set. Using this evaluation and a set of heuristics, EXPL decides which solutions should be kept to generate other solutions by yet another application of the graph transformations.

Figure 5 is the ISP description of an 8-bit multiplier that will be used as a running example to illustrate various aspects of EXPL. The algorithm is a variant of the shift-and-add algorithm. The multiplier is in the P register and the multiplicand is in the MPD register and is assumed to occupy the leftmost 8 bits of the register. The product will be in the P register. The partial products are formed in the left hand side of the P register and shifted to their appropriated position in the final product. A counter, C, is used to keep track of the number of times the basic multiplication step has been performed. Additional details about the algorithm can be found in Reference 1.

The description begins with the specification of the label for the program (MULTIPLIER). Labels are used in ISP to identify activities so that they can be branched to, or used as subroutines.

The program itself is enclosed in parentheses, and consists of two parts. The declarations and the specification of the behavior. The former are specified as a list of individual component declarations (multiplicand, multiplier/product, and step counter), using the reserved identifiers DECLARE and ERALCED as brackets. The specification of the activities of the system is given as a list of two sequential steps. The first step (C←8) initializes the counter and the second is given by a labelled (L1) block of activities. These consist of a sequence of three steps. The first one performs the basic multiplication operation; the second step decrements the counter; the third step tests the counter to see if the operation has been completed. If the value of the counter has not reached 0 then a jump to the label is indicated by using the label as an activity. If the counter is 0 then control flows out of the labelled statement and reaches the end of the program.

```
MULTIPLIER:=
(declare
    MPD<15:0>;
    P<15:0>;
    C<15:0>
eralced
C←8 next
L1:=( (decode P<0>=>
            P←P↑SRO 1;
            P←(P+MPD)↑SRO 1) next
        C←C−1 next
        (if C neq 0=>L1)
    )
)
```

Figure 5—The ISP description of the multiplier

The basic multiplication operation is described using the DECODE control operation. It implements an n-way branch depending on the value of the expression following the operator. The alternative paths selected by this operation are given as a list using the ";" as delimiter. The first path (P←P ↑SRO 1) is selected if the value of the controlling expression (P<0>) is 0; the second path (P←(P+MPD) ↑SRO 1) is selected if the value is 1. The operator ↑SRO represents a shift right inserting zeroes. The number of shifted positions is given by the second operand (in this case the integer 1).

Figure 6 shows the graph representation of the ISP description. The mapping from the ISP description to the graph form is apparent from the example. The system graph contains a unique entry point (the START operation) and a unique exit point (the STOP operation). In addition to these two operations, there can be five other types of operations in the graph model:

—branch, activates one of the output paths depending upon the value of some operand.

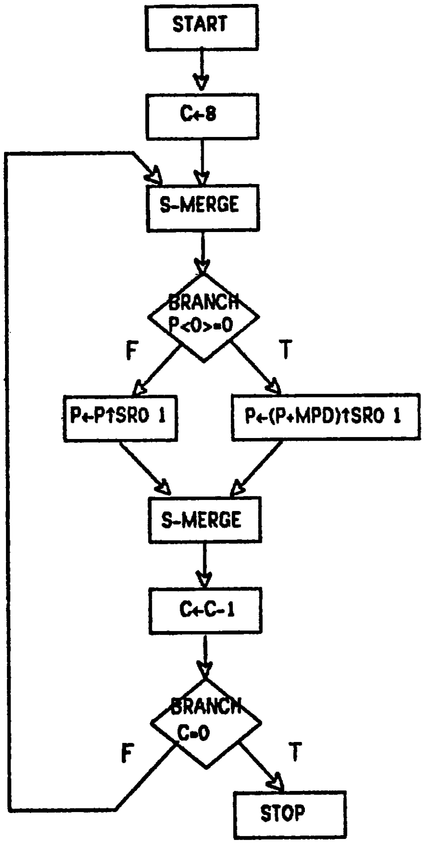


Figure 6—Multiplier graph 0 (original)

- —serial-merge, activates its output path when any of its input signals arrive
- —diverge, activates concurrently all of its output paths.
- —parallel-merge, activates its output path when all of its input signals arrive.
- —data-operation (other).

Examination of the graph for the multiplier example indicates several possible alternative designs. For instance, the computation of the loop count (C←C-1) does not depend on the shifting and adding steps (P←P ↑SRO 1 and P←(P+MPD) ↑SRO 1); the two sets of operations do not have variables in common. Thus the decrement of the loop counter can be performed in parallel with the basic multiplication step, as shown in Figure 7.

The graph thus obtained shows that the testing of the loop counter, although independent of the multiplication steps, cannot be performed in parallel with the decrement of the loop count. This fact rules out a transformation similar to the one used previously. However, it is possible to insert the testing step in the same path as the decrement. This preserves the required ordering of the counter operations but now the testing is done concurrently with the multiplication step as shown in Figure 8.

The last graph represents an "optimal" speed implementation. We are not considering, at this point, specific module-set-dependent optimizations. For instance, register allocation in RTMs or data operator allocation in MMs. This type of optimization is left up to the individual technology-dependent evaluators.

It should also be noted that, in the example above, although it took two steps to arrive at the final design we could have taken the two counter operations (decrement and testing) as one group, and place the group in parallel with the basic multiplication step. In other words, we can achieve the same final result in one step by varying the size of the graph partitions.

These graph transformations have taken us from an original solution, to two additional design alternatives. Both represent an improvement in speed with respect to the initial design. The design space, explored automatically by EXPL, can be represented as a two di-

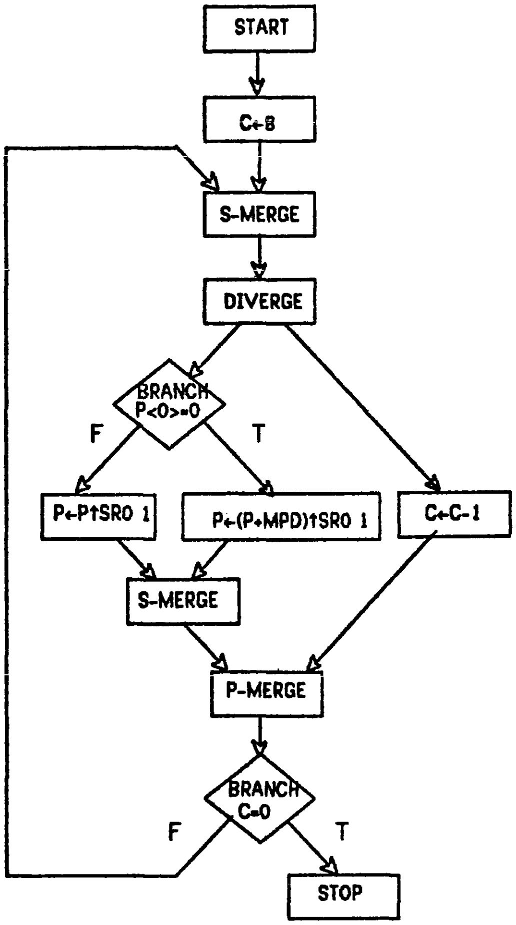


Figure 7—Multiplier graph 1

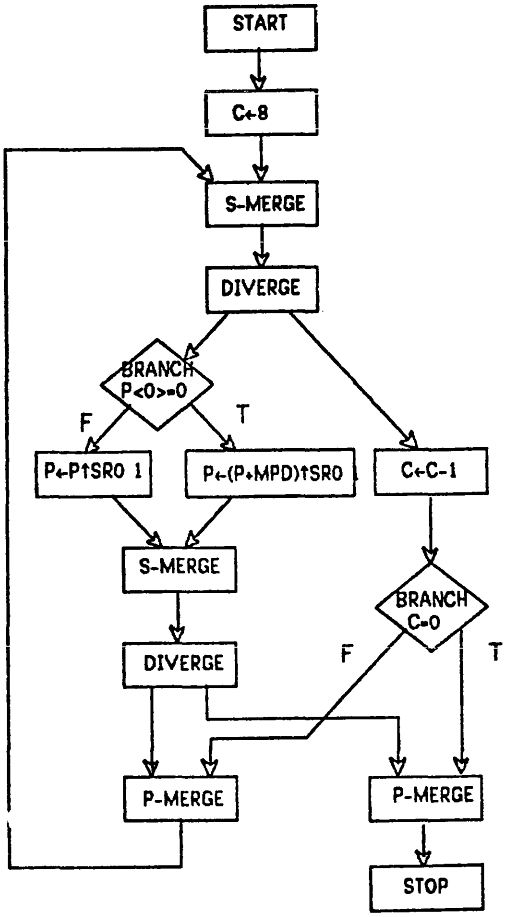


Figure 8—Multiplier graph 2

mensional plot. Each design is represented by its cost and time coordinates, computed by the technology-dependent design evaluators.

The multiplier example was implemented using both RTM's and macromodules and the design spaces are shown in Figures 9 and 10. In both plots, point 0 represents the initial solution and points 1 and 2 represent the alternative designs. Arrows indicate the steps taken in the derivation of the alternatives.

The position of the alternative design in the design space is dictated by the evaluation of its cost and speed. These two parameters are measured in terms of a specific technology, using specialized evaluator routines, as described in the following section.

MODULE SET EVALUATORS

Given a candidate design, whether it is the initial graph or one obtained by a transformation, its cost and speed must be measured to ascertain its relative position in the design space. This position indicates the quality of a solution. The evaluation process is clearly dependent on the module set used and is currently performed by ad-hoc routines, independent of the graph transformation algorithms. This section describes the evaluation procedure used for RTM[1] and Macromodule[2] systems.

The candidate design is represented by a description of its behavior i.e., a graph. In this representation the control and data operations are not bound to specific physical components. The evaluation is performed by applying a series of binding algorithms that map this (abstract) behavioral description into a physical description. This is the representation which is then evaluated. It should be pointed out that the design space we are dealing with is really an evaluation space, not a structural space. In other words, for each point in the evaluation space there may be more than one structure.

Different design spaces are explored by using either the RTM or Macromodule evaluator. The reason for this is inherent in the interconnections of the data and memory components allowed by each module set. Specifically, in the RTM set (Figure 11b) the inputs to the data operators are permanently connected to memory outputs (DMgpa). This means that some form of register allocation must be done to insure that at all times the proper operands are present at the input of the data operators. The RTM set also provides a common bus interconnection between the modules. This allows memories not directly connected to data operators to share the centralized DMgpa much as a computer's main memory shares the data operators connected to the central registers. In contrast, the Macromodule set represents a different style of design. Memory and data elements come in separate modules that are directly connected with data cables or links. Instead of sharing the data operators, each operand is bound to a register with only the necessary data operators connected to it (Figure 11a). The resulting Macromodule system has relatively expensive data operators and links for each operation, whereas an RTM system will share data operators and links (buses) with all of the operands.

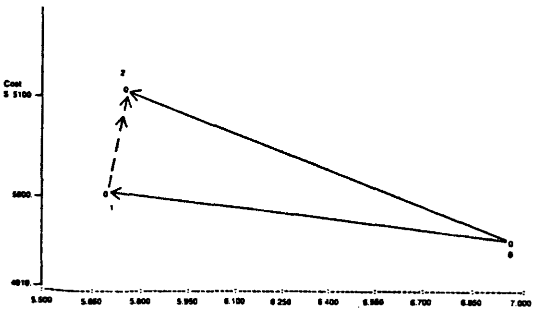


Figure 9—The multiplier design space (macromodules)

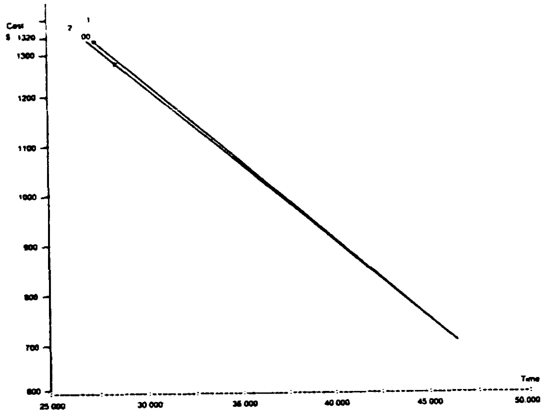
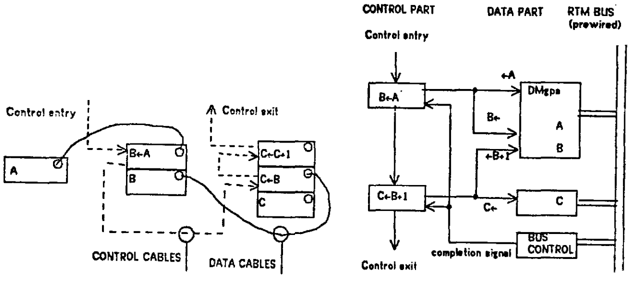


Figure 10—The multiplier design space (RTM)

Figure 12 illustrates how the concept of a *design style* is incorporated in the physical allocators of the RT-CAD system. RTM and Macromodules represent two differing register transfer level design styles:



(a) Macromodules

(b) Register Transfer Modules

Figure 11—Macromodules and RTM systems to implement B←A next C←B+1

central accumulator, and distributed data and memory, respectively. In addition, other design styles such as pipelining can be defined. An abstract system description is presented in graph form to one of the design style allocators. Each design style allocator has its own procedure for allocating operands to memories and data operations to operators. Temporary registers and extra register transfers might be added to the abstract design so that it will conform to physical design constraints. The result is a perturbed graph that reflects a specific design style and module set. The graph, which now represents a physical design, is evaluated using the costs and speeds given in the module set description. The results of the evaluation are then used by EXPL to produce an alternative design.

The allocators must be able to work interactively with the module independent subcomponents, designing and evaluating at varying levels of detail in the design process. At the style level, the allocator begins binding variables and operations to physical components using a set of allocation algorithms specific to the design style but general enough for any module set reflecting that design style. At the module set level, information specific to an individual module set is read from the module set description and used to help complete the allocation. In this way the allocator can function with several different module sets reflecting the same design style. An example of this can be found using the RTM set. Consider a module set similar to RTM except that the A and B registers of the DMgpa are symmetric. The allocation algorithms are very similar except for a few details such as not being able to produce <–B–A in one register transfer in the real RTM (although <–A–B is available).

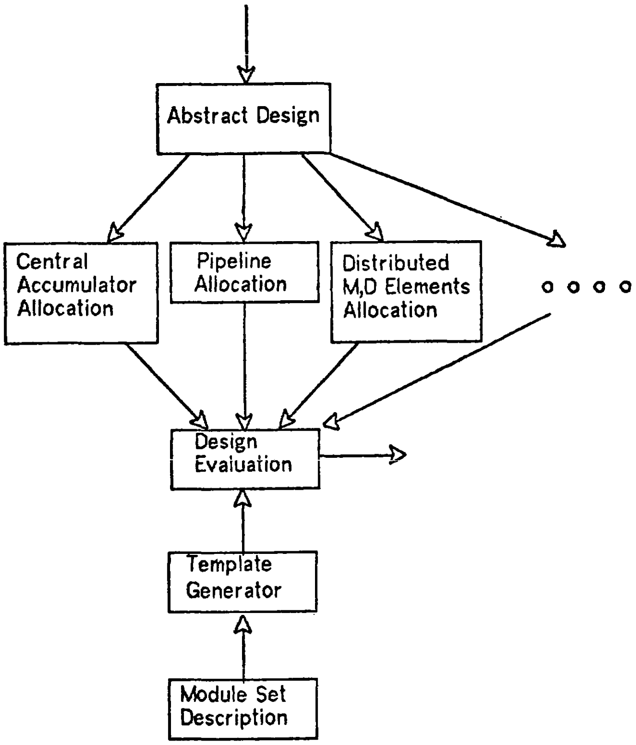


Figure 12—Module set relative design process

The module set level of the allocation uses a set of templates to map the abstract operations in the graph into a module set specific structure. These templates, produced by the template generator from the module set description, act like macros in an assembly language. For each abstract data operation in the graph there is a set of templates that indicate, for example, the different groups of RTM operations that can be executed in order to achieve the desired effect. An operation like A<–B+C, where A,B, and C have been allocated to memories without data operators (central accumulator design style allocation) is mapped into a sequence of RTM primitive operations like: DMgpa/A<–B; DMgpa/B<–C; A<–DMgpa/A+DMgpa/B (module set level allocation).

Templates of this nature are used by the Macromodule evaluator, although the nature of MMs allows more flexibility than RTMs. For instance, using the above RTM example, there are several ways of implementing the statement in MMs and the choice may be critical. With all variables allocated to registers and the necessary data operators connected between the registers (distributed data and memory design style allocation), the statement A<–B+C can be implemented alternatively as A<–B; A<–A+B, or A<–C; A<–A+B, since at the module set level of allocation it is found that all data operations have the destination variables as one of the source variables. (A module set without this restriction could use the same design style allocator.)

This decision is critical and depends upon the data operators already connected to a given register (in this case, register A) or which will be connected in order to implement subsequent operations. If the operator A<–A+C has already been placed in the stack of register A, the first option is clearly the one to adopt. On the other hand, if none of the operators exist then the template adopted depends upon the future uses of the operator, a decision based on a global analysis of the graph.

In addition, a module set may contain a data operator which cannot be described in a simple ISP statement. For instance, the RTM DMgpa can operate on two operands and then shift the result right in one register transfer. Operations such as this require that the allocator scan the graph for occurrences where the special operator can be used. Detecting these occurrences usually accounts for major optimizations in the final design. In fact, most good designers look for special features of module sets and try to utilize them wherever possible. The Macromodule allocator was written to search for special operations such as:

—Invariant bit mapping within an operand (field extraction, bit swapping, etc.)
—Comparing with constants,
—Comparing constants with bit fields,

Macromodules have special physical constructs that allow the above operations to be done easily. This specific information about the Macromodule set injected at the module set level of the allocator allows the automatically generated designs to approach designs produced by a good designer.

Table I compares some sample designs produced by hand and by the RT-CAD system.

HEURISTICS AND DESIGN SPACE TRADE-OFFS

Due to the interaction between series/parallel transformations in EXPL it is a difficult task to formalize the optimization (improvement of alternative structures) as a mathematical optimization problem. The main difficulty is the fact that transformations apply to subgraphs of arbitrary size and, as a consequence transformations in a given alternative structure may or may not be feasible or desirable in structures derived from it. It is also the case that new cases of transformations become feasible or desirable only after a specific sequence of transformations has been applied.

Two parameters will be used to describe the design space: The cost of the hardware involved and the operational time. The former is obtained by adding the costs of the components used in both the data and control structures. The latter is obtained from the average speed of the operations involved.

For a straight sequence of operations the time required is the sum of the individual times. In the presence of concurrent activities, the operation time is that of the longest (timewise) sequence.

When computing the times required by the alternative paths of a branch operation EXPL assumes, by default, that all such paths have equal probabilities of being executed (the probability being 1/n for n-way branches). This default can be overruled by the user by specifying the branching probabilities for the individual paths. The computation of the times required by the paths is then weighted by the branching probability associated with the path. The execution time is then the sum of these weighted path times.

The presence of cycles (loops) adds some complexity to the estimation of the operation time. In this case the level of nesting is assumed to be proportional to the frequency of execution of the operations. Conceptually this is equivalent to replacing the cycle by a sequence of multiple copies of the individual operations. Since the number of times a loop is executed (i.e. the number of copies) is usually unknown, a default (2) is assumed (this is a consequence of the default 50 percent probability of branching back to the loop head). This default may be overruled by the designer by specifying an estimate loop count or, alternatively, simply the branching probabilities if the loop count is not known.

TABLE I

Design		Hand Design	RT-CAD Design
Minicomputer	*Cost*	$6250	$6500
	Time	3.90usec	3.92usec
MSG	*Cost*	$6000	$7450
	Time	7.1usec	13.1usec
Multiplier	*Cost*	$3900	$3900
	Time	5.7usec	5.7usec

Having defined the parameters of the design space we can now describe the trade-offs involved in the transformation rules. Connectivity and data dependency are used in the system to indicate the feasibility of a transformation. Feasible transformations, however, do not imply necessarily any advantage in their application and the desirability of such a transformation is indicated by a different set of conditions.

The exploration of the design space in our system is performed by a group of heuristic routines that produce alternative designs in a goal oriented fashion; the goal being specified by the designer. Ideally, the goal is to find an alternative structure whose position in the design space is as close as possible to the origin (0 cost and 0 time). This ideal case is, however, not easily found in real solutions. The usual case is that the least expensive solution is not the fastest and vice versa. This characteristic provides a rough classification of the design objectives into two classes: minimal cost and minimal time.

Although a designer's aim can be classified according to these objective functions, it may be the case that the real objective is more complicated in nature, namely, some combination of time and cost. For instance, the objective could be something like: "the fastest alternative structure not costing more than x dollars".

For simplicity, the subspace of acceptable solutions will be defined by a set of straight-line segments whose slopes reflect the objective functions. In the example above a single straight line, parallel to the cost axis, would be used to divide the space in two halves. Only those solutions that lie in the semispace containing the origin are considered acceptable. These solutions represent improvements along the design goal.

More complex constraints can be described by using lines of the form $C=m*T+b$, where m is a parameter indicating how many dollars the designer is willing to pay for each time unit saved (if time is the primary goal) or how many time units the designer is willing to sacrifice for each dollar saved (if cost is the objective). m is termed the cost/speed trade-off factor. An example, Figure 13, will clarify this description.

Assume that the primary objective is a reduction in time and that the designer wants a cost/speed trade-off of at most m dollars for each time unit improvement. Furthermore, assume that the original design is characterized by C1 and T1. The "acceptable trade-off" subspace would thus be delineated by two line segments: one parallel to the cost axis starting from

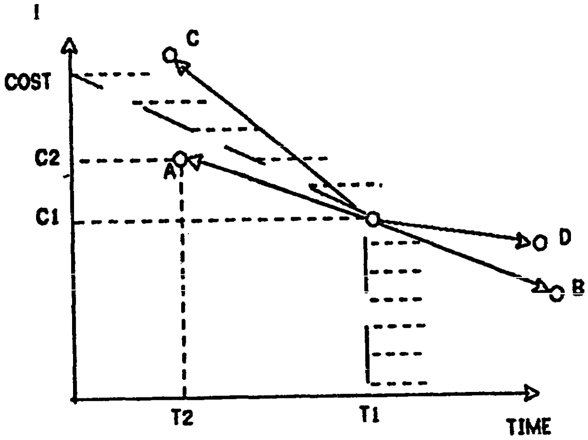


Figure 13—Design space reduction

(T1,C1) to (T1,0), and the other through (T1,C1) with slope −m. Further assume that by studying the control flow and data dependencies in this original structure, four transformations are available which yield four alternative solutions derived from the original one: A,B,C,D.

By dividing the space according to the trade-off lines alternatives B, C, and D can be rejected because their characteristics are not within the acceptable subspace (i.e., they take more time or the decrease in time costs too much). The alternative left, A, represents improvement in time while the cost to achieve the improvement is under the cost/speed trade-off threshold.

The process can now be applied to A in an identical manner. Design A is taken as the new initial solution and a new "acceptable trade-off" subspace is defined by a line segment (T2,C2) to (T2,0) and a line with slope −m through (T2,C2). Since in some cases more than one alternative can be left for further exploration, this process takes the form of a tree walk where the nodes represent alternative solutions and the edges are the transformations applied. In some instances, identical structures can be obtained by different sequences of transformations and the exploration of the design space is a graph-walking process. In any event, a path ends when no alternative solutions worth exploring can be reached from a given point. When all possible paths have been explored the end nodes are measured against the primary objective and the best one chosen.

In general, the space of alternative solutions looks more like a graph than a tree. Several paths (i.e., sequences of transformations) may lead to the same solution.. Thus, it is important to detect points in the space that have already been examined. Other problems that arise in the exploration process have to do with the cost of the process itself. EXPL does not perform a brute force search. Accepting an alternative solution for further exploration depends upon the goals indicated by the user. Besides the main goals (speed, cost, and a trade/off factor) mentioned before, the user can also specify a minimum percentage gain for a transformation-derived solution to be acceptable. This is called the improvement factor. If the gain falls below the improvement factor, the new design is rejected.

This pruning process, when applied indiscriminately, can lead to an incomplete exploration. It may be the case that, although a derived solution is worse (according to the goals) than its parent solution, solutions derived from the former could in fact be better than the parent. EXPL handles the detection of this type of local optimality by allowing the user to specify a rejection level. The rejection level indicates whether or not non-improving solutions are to be further explored. The user specifies the maximum length of such non-improving paths.

The following section briefly presents several examples of design spaces. The examples illustrate some of the points discussed previously.

SAMPLE DESIGN SPACES

In this section we will present three examples of the design spaces explored by EXPL. We will not discuss the specific systems whose design spaces are depicted in Figures 14, 15, and 16. The examples will be used to show the characteristics of the design spaces and the exploration procedures.

Figure 14 shows the design space for a RTM system that is used as a controller for the X- and Y-plates

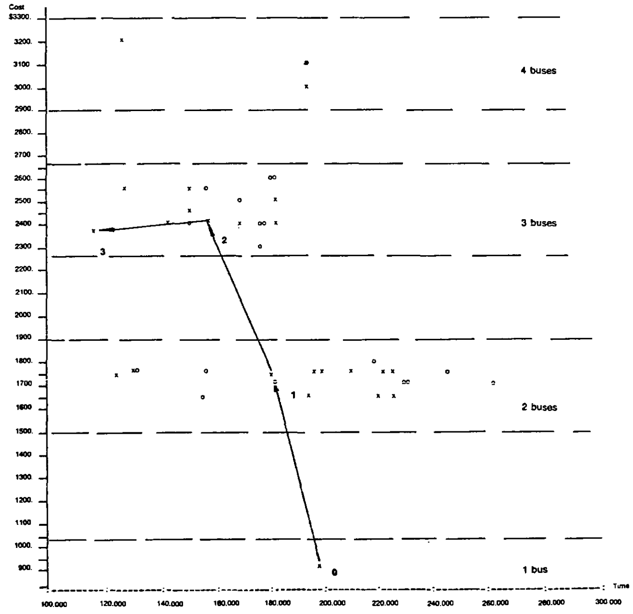


Figure 14—RTM design space (MSG system with 10 percent improvement factor)

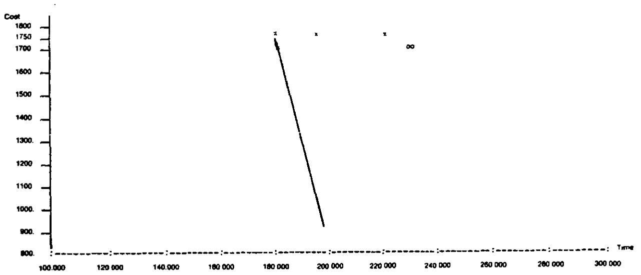


Figure 15—RTM design space (MSG system with 20 percent improvement factor)

of an oscilloscope. The system is used at CMU for RTM demonstrations (the "Munching Squares Generator"). Evaluations of individual designs are represented by a cross. A circle denotes that more than one design had the same evaluation. The first characteristic that can be noticed in this evaluation space is the stratification of the alternative designs. The solutions appear in horizontal bands representing solutions of similar cost. This is due to the high cost of the RTM buses compared with the cost of the other modules in the RTM set. The space is divided into bands corresponding to the 1, 2, 3, and 4 bus solutions.

The figure shows the degrading effect in RTMs of sharing variables between concurrent computations. The best solution (in terms of speed) used 3 buses and is faster than the 4 bus solutions. The algorithm is such that, although it allows a high degree of concurrency, when this degree exceeds a certain threshold there is a loss of speed in the total system due to the overhead of establishing the concurrency. The path followed to find the best solution is indicated in the figure. It is interesting to observe the transition from solution 2 to solution 3. There is a substantial gain in speed together with a reduction in cost. The explanation is that once the cost of a bus has been accepted as a reasonable price to pay for a given gain in speed it does not cost much to spread the load and perform more operations concurrently. Indeed, as the example shows, alternative allocation of the computations to the buses, for a fixed number of buses, is crucial.

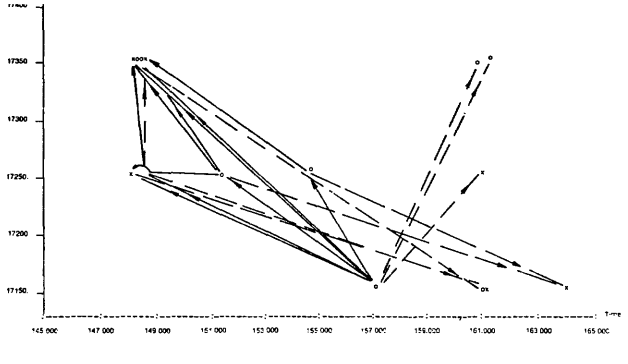


Figure 16—Macromodule design space (conveyor-bin system)

When a solution is analyzed the set of feasible transformations that can be applied to its graph is tabulated. The improvement factor specified by the designer is then used to prune this table. This pruning takes place before a transformation is applied and is based on a preliminary "best case" analysis of a candidate transformation. The solution derived by applying the transformation may or may not realize the potential gain indicated by the preliminary analysis. This reduction in the predicted gain is due to several causes. If the goal is a reduction of cost, performing two concurrent operations in sequence may not in the case of RTMs result in a reduction in the number of buses (other computations may require the bus that was thought to be expendable). If the goal is a gain in speed, adding buses may result in a loss of speed due to the time required to copy and move variables between the buses in the system. Similar considerations can be applied to the case of Macromodules.

Figures 14 and 15 correspond to the design space for the same RTM system explored using the same cost/speed trade-off factor ($100/microsecond) but using different improvement factors. In the space shown in Figure 15, the preliminary improvement factor was set to a higher level (20 percent) than in the space shown in Figure 14 (10 percent). An interesting phenomenon occurred. The transformation indicated by the directed line in Figure 15 had a very promising preliminary evaluation (over 30 percent predicted gain). When the transformation was applied, the new solution did not realize the predicted gain. It was, nevertheless, better than the original solution and was later chosen by the system as the best solution. All feasible (i.e., applicable) transformations to this new solution were then examined and none of them promised to be better than the improvement factor. All of these transformations were rejected and the exploration path was terminated. When the same situation appeared in the example of Figure 14, there were several transformations that were better than the new, lower, improvement factor. One of them led in fact to the best solution of the space of Figure 14. It is interesting to observe in Figure 14 that the slope of the transformation from solution 1 to solution 2 indicates a better cost/speed trade-off than the transformation for the original solution (point 0) to solution 1. The gain in speed produced by the transformation, although smaller than the improvement factor used in Figure 15, was achieved completely; there was no overhead added to the system by the extra concurrency.

This type of anomaly is not uncommon in the modular design spaces explored so far, if anything, they tend to be the rule rather than the exception. The pruning of the applicable transformations, based on a preliminary analysis, can lead us to ignore certain transformation paths that may yield better solutions. It is valid to ask, "why then should the system do any pruning at all?" The only reason we can provide is based on the analysis of the cases studied so far. Applying a transformation without any considerations to its possible gain is an expensive proposition. For any solution, branching factors (i.e., number of feasible transformations) of 30 to 50 are not uncommon. Applying a transformation implies a reconfiguration of the graph and the recomputation of several associated tables—an expensive operation in the current implementation. Applying each feasible transformation can lead to a very expensive design process. For example, when EXPL explored the design space of Figure 14, 454 transformations were analyzed. Of these, 337 were deleted for not meeting the improvement goal. The resulting 117 transformations (26 percent of the original) were applied to obtain new designs. Of these, 87 were deleted for not meeting the cost/speed trade-off or for being duplicated designs ("loops" in the design space exploration). As a result, only 30 transformations (6.5 percent of the original) produced designs that were improvements over their predecessors. For large systems, the combinatorial explosion of possible designs is prohibitively expensive. More intelligent heuristics are required to home in on the most interesting parts of the design space. Once these areas are located a near exhaustive search can perform the final selection. One such intelligent heuristic would be to identify the loop structure of the algorithm and apply transformations to the innermost loops. A variation of this might be transforming those loops whose (improvement factor) × (cycle count) product is largest. Furthermore, loop control and loop computation are parallelizable in many circumstances and their identification could speed up the transformation search which is seeking to identify any potential parallelism. Research into better heuristics is being actively pursued.

The system as implemented allows the designer to guide the exploration via an interactive command language. In the interactive mode, EXPL does not perform any pruning and the designer is free to order the system to perform any feasible transformation, regardless of its predicted gain. The automatic mode of exploration can therefore be used selectively under user guidance, assisting the application of the heuristics.

Figure 16 shows the design space for a system designed as a controller for a conveyor-bin unit. The design space corresponds to the alternative designs implemented using Macromodules. The figure is a good example of a design space with multiple paths leading to the same solution. The space configuration also indicates the characteristic of Macromodular systems. Once a basic design is implemented, variations in the level of concurrency do not present the radical changes in cost typical of RTM system. The basic costs of the Macromodular system are given by the memory and data operation modules (the "stack"). Variations in concurrency only imply adding or eliminating control modules and cables, a minor fraction of the total cost.

VAS AND DESIGN VERIFICATION

It is possible to develop an ISP description that is syntactically correct but that does not make sense logically. Figure 17.a depicts a syntactically correct ISP while Figure 17.b illustrates the corresponding graph. The graph is essentially the same one produced by the ISP compiler. The data operations have been deleted as a notational convenience (we can think of the data operations as being assimilated into the arcs connecting the control operations).

In the case of x=1 the right half of the parallel merge in the graph would receive two control signals (one from the right half of the diverge, the other from the left half via the branch). The other input to the parallel merge would not receive a control signal and the system would deadlock at the parallel merge. Analytical tools based on the vector addition system (VAS)[10] have been programmed to detect such design flaws.

The VAS is best introduced by example. Consider Figure 17.b. The arcs in the graph represent register transfers while the vertices represent control primitives. Each arc may contain tokens representing evocation of the associated register transfer. Graphically a token is represented by a dot on an arc. A marking of a graph with r arcs is a mapping from the set of r arcs to an r-dimensional vector of nonnegative integers, each of which represents the number of tokens on the corresponding arc.

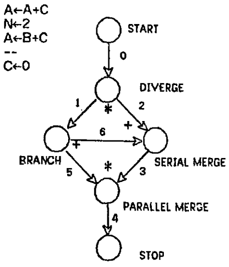
A vertex with a token on its single input arc is said to be enabled. Only enabled vertices can fire. The firing of a vertex removes a token from the input arc and deposits a token on its output arc. For the case of multiple input arcs there is an associated logic condition, either disjunctive (signified by a+) or conjunctive (*). A vertex with disjunctive input arcs is enabled when any input arc has a token. Firing the vertex removes a token from one input arc. This corresponds to a serial merge in the compiler producer graph. A conjunctive input condition requires tokens on all input arcs before the vertex is enabled (parallel merge). Firing the vertex removes a token from all the input arcs. Likewise a set of output arcs can be disjunctive or conjunctive. When a vertex with disjunctive output condition fires it places a token on one of the output arcs (branch). The conjunctive condition places a token on all the output arcs (diverge). A simulation is a sequence of permissible vertex firings.

Simulations are conveniently represented by the Vector Addition System (VAS).[10] Figure 17.c depicts the VAS for the graph in Figure 17.b. The VAS con-

```
TEST := ( declare A<15:0>; B<15:0>; C<15:0>; X<>; N<15:0> [illegible]
        (N←2 next LABEL := A←B+C);
        (A←A+C next (decode X=> C←0; (C←1 next LABEL)))
       )
```

(a) ISP Description

Arc	Operation
0	--
1	A←A+C
2	N←2
3	A←B+C
4	--
5	C←0



(b) Graph Model Illustrating Deadlock

D-Index	Associated Vertex in (b)	Displacement Vectors
D1	Diverge	-1 1 1 0 0 0 0
D2	Serial Merge	0 0 0 1 0 0 -1
D3	Serial Merge	0 0 -1 1 0 0 0
D4	Parallel Merge	0 0 0 -1 1 -1 0
D5	Branch	0 -1 0 0 0 1 0
D6	Branch	0 -1 0 0 0 0 1

Initial Marking M0 = 1 0 0 0 0 0 0

arc 0 ↑ arc 6 ↑

(c) The Vector Addition System for the Graph in (b)

Figure 17

sists of an initial marking vector Mo and a set of displacement vectors which correspond to vertices. Each component of the vector corresponds to an arc. All valid firings (new markings) of the graph can be determined by adding a displacement vector to the current marking Mi. Those additions which result in all marking vector components being nonnegative are valid markings and can be used to establish subsequent valid markings. For example, the only valid marking from the initial marking Mo resulting from the addition of a single displacement vector (e.g., D1) in Figure 17.c is (0,1,1,0,0,0,0,). The displacement vector D2 does not lead to a valid marking since the result of its addition to M0 is (1,0,0,1,0,0,-1).

A control flow tree depicting all possible markings (or states) of the VAS can be constructed. A portion of that tree for our example is shown in Figure 18. Nodes are appended to the tree until, for each leaf, either its marking is identical to that of one of its ancestors or no displacement vectors can be applied. In either case the node is called a leaf.

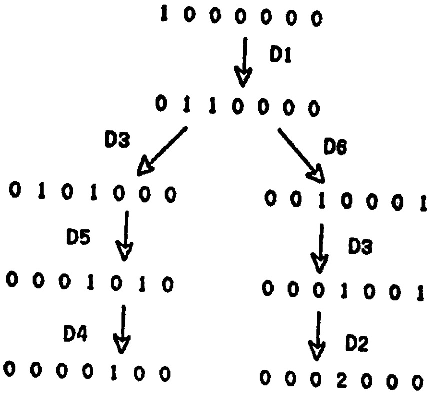


Figure 18—Portion of the control flow tree for the example of Figure 17

Properties of this tree can be used to detect properties of the graph. For example, the leaf (0,0,0,0, 1,0,0) represents a properly terminating sequence since there is a single token on arc 4. By contrast, leaf (0,0,0,2,0,0,0) represents two tokens on arc 3. No tokens are on the exit arc. This is the deadlock situation alluded to earlier. Furthermore, depending on the actual physical implementation of the graph, this leaf may indicate a lost signal.

OTHER DESIGN TOOLS

One of the traditional design automation tools has been the simulation of a digital system. The RT-CAD system includes a simple simulator (SIMU10) which interfaces with the machine descriptions stored in the data base. Data base objects consist of two components: a symbol table and a flowchart. The former contains the description of the digital system component (registers, flags, memories, constants, etc.). The latter contains the data and control operations describing the behavior of the system. By stepping through the flowchart, SIMU10 can emulate the behavior of any digital system stored in the data base. An interactive command language allows the user to set and display the contents of the registers and memories of the system. The command language allows the user to set and reset arbitrary breakpoints and to interpret command files, created off-line, instead of prompting the user directly.

It is also desirable to produce designs according to criteria other than the traditional cost/speed criteria. One such criterion is testability. The structure of the final design substantially determines the ease with which tests can be generated for the design. A testability measure[11] has been developed that correlates well with actual test generation effort. It is important to note that the common representation used as input to the various design programs is a critical feature that insures that the algorithm being evaluated is actually the one being implemented, verified, or simulated by the other design programs.

It is worth mentioning that some of the RT-CAD system components have been exported and are being used in places other than CMU. One of the more ambitious projects is the Computer Family Architecture (CFA) being developed by the Naval Research Laboratory. The objective is the implementation of a generalized Architectural Research Facility (ARF). NRL is currently using the ISP compiler and SIMU10. The ultimate goal is the implementation of a system that will emulate large machines described in ISP. Interactive facilities will assist the user in evaluating the described machine. Thus, proposed machines can be evaluated and their architecture tuned prior to production. The emulator will be entirely resident in the control storage of a dedicated PDP-11/40E (a commercially available, modified PDP-11/40 that allows the coexistence of user written microcode together with the standard, DEC provided microcode that implements the PDP-11 instruction set).

FUTURE DIRECTIONS

To achieve the goal of automatic design relative to technology a mechanism is required that would take the description of a module set and create the equivalent of the ad hoc module set evaluators currently in use. It was also noted earlier that the order of physical allocation (registers, buses, operators, etc.) is a strong function of the design style imposed by the module set. This information would also have to be extracted from the module set description.

This preliminary design automation system and a machine relative optimizing compiler-compiler project serve as a stepping stone to an even more ambitious project termed the Symbolic Manipulation of Computer Descriptions (SMCD)[14] depicted in Figure 19. There is a continual stream of new machines spurred by the advent of minicomputers and microprocessors. Each machine has a different instruction set. The emergence of microcoded systems with the option of user-defined instruction sets has increased this flow of instruction sets. Each new system requires supporting software and the amount of software grows for any individual system as user requirements grow.

One direction in which to seek a solution to ease the burden of software development is to relativize the production of software to the description of the machine. The central ingredient of this approach is the description of computer systems in a symbolic form, such that a range of problems can be solved by manipulation of these descriptions.

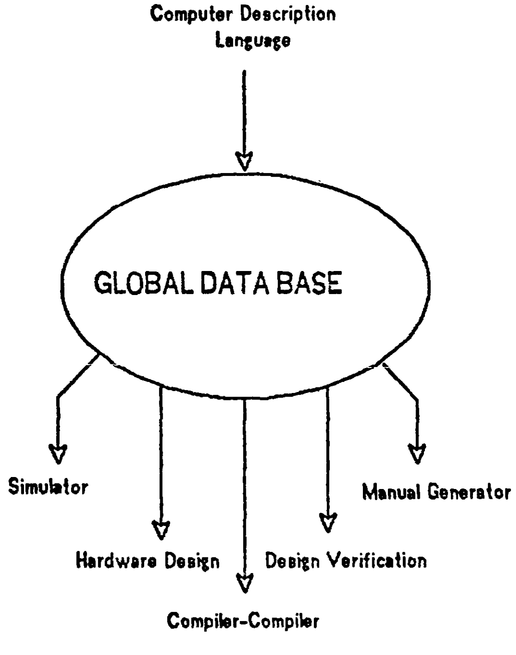


Figure 19—The symbolic manipulation of computer descriptions

Figure 19 depicts the scope of the SMCD project. The ultimate goal would be to produce and evaluate a computer system from its behavioral specifications, together with the documentation and system programs. Thus the delay from the conception of a new architecture to the time it is implemented and ready for users can be significantly reduced.

ACKNOWLEDGMENTS

We would like to thank the many individuals whose work contributed to the results described in this paper. Steve Goldman, Ross Scroggs and Phil Karlton assisted in the development of the ISP compiler. Vittal Kini implemented the VAS, Don Thomas implemented the MM evaluator, and Steve Rodkey implemented SIMU10.

REFERENCES

1. Bell, C. G., J. Grason and A. Newell, *Designing Computers and Digital Systems,* Digital Press, Digital Equipment Corporation, 1972.
2. Clark, W. A., "Macromodular Computer Systems," *AFIPS Conference Proceedings,* Vol. 30, SJCC 1967, pp. 335:402.
3. Breuer, M. A., "Recent Developments in the Automated Design and Analysis of Digital Systems," *Proceedings of the IEEE,* Vol. 60, No. 1, January 1972, pp. 12:27.
4. Darringer, J. A., *The Description, Simulation and Automatic Implementation of Digital Computer Processors,* PhD Thesis, Department of Electrical Engineering, Carnegie-Mellon University, May 1969.
5. Friedman, T. D. and S. Yang, "Methods Used in An Automatic Logic Design Generator (ALERT)," *IEEE-TC,* Vol. C-18, No. 7, July 1969, pp. 593:614.
6. Mesztenyi, C. K., *Computer Design Language. Simulation and Boolean Translation,* Technical Report 68-72, Computer Science Center, University of Maryland, June 1968.
7. Rozenberg, D. P. and R. L. Savage, "A Proposal for the Computer Design Process Based on Multi-Level Simulation," *IFIPS Congress,* 1971.
8. Digital Equipment Corporation, *PDP-16 Computer Designers Handbook,* 1971.
9. Hill, F. J. and G. R. Petersen, *Digital Systems: Hardware Organization and Control,* Wiley, New York, 1973.
10. Huen, W. H. and D. P. Siewiorek: "Intermodule Protocol for Register Transfer Level Modules: Taxonomy and Analytic Tools," *Proceedings of the Second Annual Symposium on Computer Architecture,* Houston, Texas, February 1975.
11. Stephenson, J., *A Testability Measure for Register-Transfer Level Digital Circuits,* PhD dissertation, Electrical Engineering Department, Carnegie-Mellon University, 1974.
12. Bell, C. G. and A. Newell, *Computer Structures, Readings and Examples,* McGraw-Hill Book Company, New York, 1971.
13. Barbacci, M. R. and D. P. Siewiorek, *Some Aspects of the Symbolic Manipulation of Computer Descriptions,* Department of Computer Science, Carnegie-Mellon University, 1974.
14. Barbacci, M. R. and D. P. Siewiorek, "Automated Exploration of the Design Space for Register Transfer Systems,' *Proc. of the First Annual Symposium on Computer Architecture,* University of Florida, Gainesville, ACM SIGARCH, Computer News, Vol. 2, No. 4, December 1973.

Some computer-related advancements for enhancing U. S. shipyard productivity

by RICHARD B. WISE and DOUGLAS J. MARTIN
IIT Research Institute
Chicago, Illinois

ABSTRACT

An exciting new cooperative program for enhancing productivity in the shipbuilding industry through improved automation technology is described. This program, designated as the REAPS (Research and Engineering for Automation and Productivity of Shipbuilding) Program, entails joint efforts of the Maritime Administration and the U.S. shipyards toward new capability development.

Four on-going hardware/software-related projects currently under way are described in some detail. These include: a major software system (AUTOKON-71) for design and construction of ships; a specialized CAD/CAM remote batch graphics terminal to better serve shipyard needs; an interactive graphics system for pipe detailing; and a minicomputer-based numerically controlled frame bender.

A number of other hardware/software development programs currently approved but not as yet funded are also discussed.

THE NEED

Over the years a decided productivity lag has developed between U.S. shipbuilders and other shipyards around the world. For instance, a few years ago it was shown that U.S.-built container ships typically required 120 man-hours per delivered ton, while the equivalent ship built in West Germany required 90 man-hours per delivered ton. For ore carriers the U.S. required 75 man-hours per delivered ton whereas Japan required 59 man-hours per delivered ton.

Until 1970 the only recourse available to the U.S. Government to keep our Merchant Marine facilities competitive was to provide a subsidy program which reimbursed ship operators and ship builders the difference between the cost of materials and labor in the U.S. and those of competitive foreign countries.

In 1970, however, new laws were enacted which set in motion a large-scale research and development program in cooperation with U.S. shipbuilders and ship operators aimed at reducing costs and to improving productivity rather than merely reimbursing inefficiency.

THE SOLUTION

As a first step toward administering this program for the shipbuilding industry, the U.S. Maritime Administration convened a group of industry experts to identify where the most critical deficiencies existed and to provide guidance as to the best means for resolving them. The recommendations of this Technical Advisory Group resulted in the formulation of a number of MarAd supported research and development activities addressing six major development areas: computer automation, welding technology, surface preparations and coating, materials handling, ship producibility, and general projects.

In an effort toward developing a coordinated thrust in administering the resulting programs for applying automation and computer technology to shipbuilding, the REAPS Program was born. REAPS, an acronym for Research and Engineering for Automation and Productivity in Shipbuilding, is a cooperative program involving the U.S. shipyards and the Maritime Administration in a total system approach designed to identify and take advantage of productivity opportunities through the application of automation technology. Developments associated with this program are keyed to specific applications. Projects are not considered complete and successful until they have been implemented under shipyard production conditions. This implementation phase precludes the possibility of a project failing because of poor implementation of a sound development. It also insures that only those projects with valid objectives will be undertaken.

The REAPS program itself is funded jointly by MarAd and the participating shipyards and is administered by a contractor (IIT Research Institute). Its activities entail three major efforts: (1) Advance planning to recognize future productivity opportunities; (2) Library and information services to apprise the industry of the latest available information on technology; and (3) R&D Program formulation.

The advance planning activity involves the identification of high-cost areas and subsequent target opportunities for new hardware and software research and development efforts.

The library and information services activity involves: (1) the quarterly publication of a Shipbuilding Technology Update Bulletin containing selected up-to-date citations and abstracts on world-wide publications of interest to shipbuilders; (2) maintaining a library of shipbuilding technology information and selected software programs; (3) distributing a library catalog and copies of the Update Bulletin to all major U.S. shipyards and providing a document reprint and software distribution service; and (4) holding an annual REAPS Technical Symposium where industry leaders are invited to discuss new advances in shipbuilding automation.

The R&D program formulation activity entails both project initiation and project monitoring. The advance planning activity identifies opportunities for productivity enhancements. Representatives from the REAPS yards then collectively prioritize these indicated requirements, develop project briefs defining detailed needs, solicit proposals from competent agencies for fulfilling these needs, and determine the best source for their realization.

The resulting development projects are then carried out on a cost-sharing basis with MarAd funds, with monitoring of all progress being pursued under the cognizance of the REAPS program.

ONGOING PROGRAMS

The remainder of this paper will be devoted to detailed discussions of the extant and contemplated computer-related projects within the REAPS Program.

AUTOKON SUPPORT PROGRAM

Background

The AUTOKON Support Program is actually the charter REAPS activity. Its genesis stemmed from actions taken by a U.S. shipbuilding industry advisory group in 1971 in response to a recognized void within the industry in the area of computer-aided shipbuilding. This requirement was most pressing for hull fairing functions and for numerically controlled burning of steel plates.

Accordingly, to resolve this deficiency, the Maritime Administration, acting on the advice of this Technical Advisory Group, purchased U.S. rights to a software system developed by a Norwegian Shipbuilding firm, the Akers Group. This system, known as AUTOKON-71, was in active use in more than 40 European shipyards.

MarAd then sublicensed AUTOKON use rights to U.S. yards. The AUTOKON Support Program, as it is now called, was then established to provide a mechanism for adapting this system to the U.S. shipyard environment.

A detailed discussion of the AUTOKON-71 system itself is contained below. Following that is a discussion of the activities of the AUTOKON Support Program for effecting enhanced productivity through cooperative efforts in adapting the system for U.S. requirements.

AUTOKON features and structure

Overall organization

The overall organization of the AUTOKON-71 system is shown in Figure 1. The system itself is completely modular, thus allowing shipyards to pick and choose among the modules to best serve their individual needs, with the ability to easily substitute alternate programs if they desire.

As shown in the figure, the entire system is tied together through the use of a common database. The relationship of each of the AUTOKON subsystems to this database will be elaborated below. In general, however, all new data generated by the modules of each of the subsystems are accumulated within the database and any subsequent module may reference previously-generated data from the database. Thus it is possible to gradually accumulate the information de-

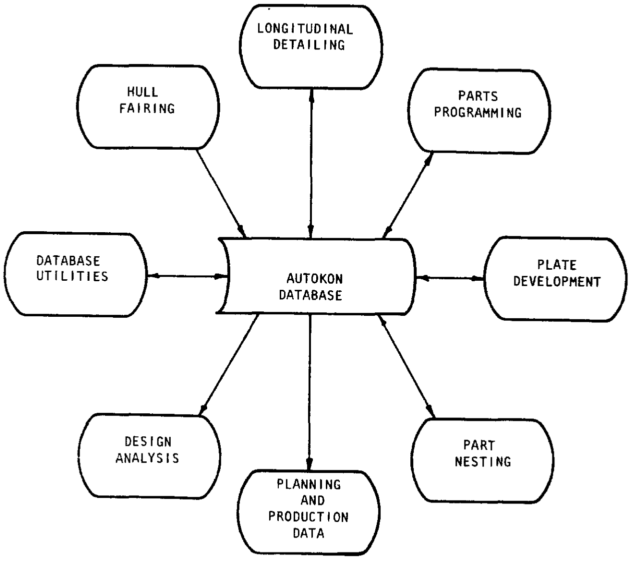


Figure 1—The Autokon-71 system

fining a ship, in effect allowing it to be constructed within the database prior to its actual erection.

In addition to the usual economies in data maintenance and storage costs resulting from the minimization of data plethoration, further benefits are achieved in the design and construction cycle since all personnel using the system have access to identical information for consistency and efficiency.

With two exceptions (Modules ALKON and DUP—see below) each of the modules comprising the various AUTOKON-71 subsystems are applied in a simple straightforward manner involving submission of a minimal amount of data in fixed-field format on preprinted data sheets. Following the commands and information on the input sheets, additional information, as required, is automatically retrieved from the database, appropriate computations are carried out, and results are recorded on the database and output listings, with graphical plots and/or N/C tapes also produced as necessary. The N/C tapes may be input to a drafting table, flame cutter, or alternatively, via a postprocessor, to a Tektronix CRT display or Calcomp drum plotter for quick-look review.

The two exceptions alluded to above are equally simple to apply but incorporate an interpretive command language in lieu of the fixed input format.

The entire system is structured to effectively fit within existing working conditions at a shipyard, with the overlying philosophy assuming that the users of the system are subject knowledgeable but have no background in computer processing.

Except for a handful of assembly-language programs for handling computer-dependent functions and inner-loop subroutines, the entire AUTOKON-71 system is written in Fortran. Thus, the system is almost completely computer-independent, with functionally identical versions operating on Univac 1100, IBM 360/370, Honeywell 6000, and CDC Cyber 70 systems.

A detailed discussion of the AUTOKON-71 database and each of its subsystems follows.

Database management subsystem and associated utilities

The database itself is physically assigned to direct access storage (drum, disc, etc.) and accumulates to the order of two million computer words for a typical ship, such as a large tanker or container ship. Data are stored in an arbitrary number of variable-sized indexed-sequential files each residing in one or more fixed-size physical blocks as required. Every file contains a number of variable-sized logical records (designated as "matrices") each incorporating information of a specified type and having a specific format.

Files are addressed using a six-integer identifier. The meaning associated with each of these six integers is established according to a fixed set of conventions, which for each shipyard, are adapted to best adhere to local procedures and techniques.

The AUTOKON-71 database access and control subsystem is embedded in a module designated as the AUTOBASE module. These AUTOBASE routines, which are completely invisible to the user, provide an efficient access method for operating on the database. Whenever a specific six-integer identifier is referred for storage or retrieval (see Figure 2), a hashing algorithm is applied to the numbers to identify a unique page in the file catalog containing information on that identifier. That one page of the file catalog is then searched sequentially to find the most recent version of the desired file (grandfather versions of all files are optionally retained in the database to allow backup). The matching entry in the file catalog then contains a direct-access pointer to the desired file. The file, in turn, contains a File Table of Contents for accessing the desired matrix.

Thirteen independent utility modules are provided as an aid in administering the database. Nine of these are used to facilitate establishment and modification of the database, and three others for storing and retrieving data directly to and from the database, bypassing the other modules—useful for transferring data from N/C tape images, from a digitizer or to or from another database.

The last utility module, designated as DUP (**D**atabase **U**tility **P**rogram) is a stand-alone system for manipulating the database contents, for creating magnetic tape backup, for copying specified portions of the database onto another database, and for generating an annotated table of contents for the database. Unlike the other modules, DUP is an interpretive system using a set of commands specified free-form on punched cards. Typical commands include: READ, WRITE, DELETE, CONTENTS, COMPRESS, DUPLICATE, etc. Each command is usually followed by one or two parameters describing the data affected.

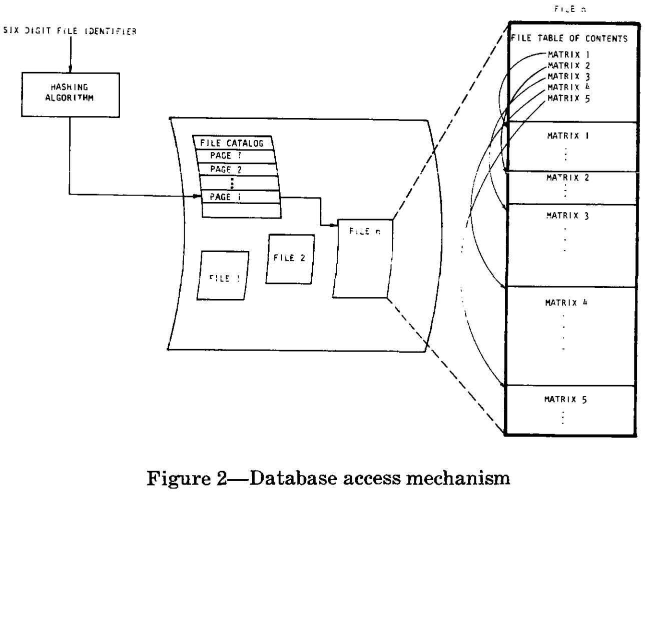


Figure 2—Database access mechanism

Hull fairing subsystem

The AUTOKON-71 hull fairing subsystem is actually comprised of three modules: FAIR for the actual fairing of the ship's lines; DRAW to generate N/C drafting table drawings of the faired lines; and TRABO for **TRA**nsferring the final **BO**dy plan to the AUTOKON database. As shown in Figure 3, the fairing process is carried out using a sequential file for storing the offset information. Because the fairing activity involves an iterative man-machine process, significant economies can be effected by performing the initial fairing runs independent of the database, incorporating only the final faired lines into the database.

The purpose of the FAIR module is to perform the fairing of ship lines and to transform these lines into a numerical form suitable for further computer processing. The basis for performing the fairing process is a preliminary lines drawing in a suitable scale. Normally 200-400 data points for an entire ship are lifted from the lines drawings, the number depending upon the extent of shape incorporated in the hull form.

Input consists of some key curves and selected points on frames, waterlines and/or buttocks. The fairing is performed automatically from the data points selected by the user, and results are presented in printed form and, via the DRAW module, in N/C form for subsequent drawing on an N/C drafting table. Drawings are automatically produced, as desired, for all curves faired.

In use, FAIR may be employed as desired to generate an entire ship or as little as one-fourth of a ship. The different parts of the ship may then be "hooked up" to form the complete ship. As any of these hull representations is satisfactorily developed, it may then be transferred from the working storage used by FAIR into the database for the desired hull and added to whatever previously exists. This body plan transfer is accommodated through the use of the TRABO module.

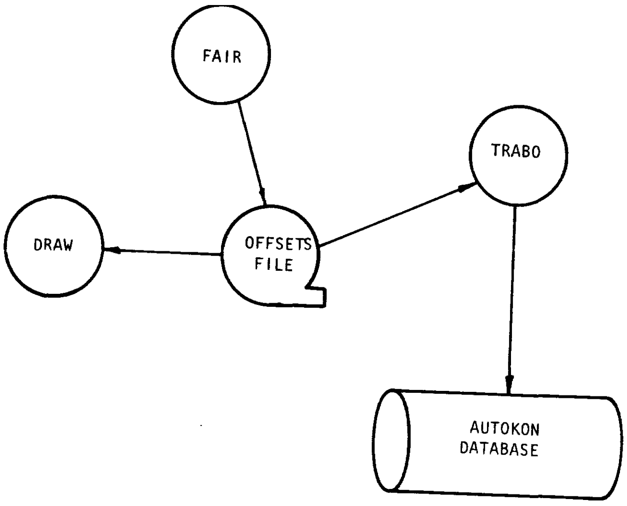


Figure 3—Hull fairing subsystem

Longitudinal detailing subsystem

Once the hull data are incorporated within the database, this subsystem can then be applied to strake the shell by fitting longitudinal curves to the hull surface. Using a module called LANSKI (a Norwegian mnemonic standing for longitudinal details), contours for seams, longitudinals, stringers, bulkheads, decks, etc. can be automatically defined.

Input to the program consists of a few points for each seam and for each longitudinal from its start point and end point, together with a simple identification of desired cutouts and of the first and last transverse frames along the shell to which the cutouts are to be applied (also using INCLUDE or EXCLUDE operators to identify explicit frames where cutouts are to be omitted).

As output, a Table of Details is produced, both in the database and as a tabulated listing, describing how and where the longitudinals, seams, and decks cut through transverse frames, bulkheads, etc. The space angle between two intersecting parts is computed and stored to provide automatic correction of the nominal dimensions for clearance, on height, width, and, if necessary, on the thickness and angle of flange, etc. Plots of the body plan with the longitudinal curves superimposed, and other drawings for checking purposes are produced.

Parts programming subsystem

The subsystem for parts detailing, designated as ALKON (standing for **AL**gorithmic **CON**struction), is by far the most sophisticated element of the AUTOKON-71 system. Input for this module is specified in a very powerful Fortran-like problem-oriented language. While it is primarily a plane geometry definition tool, ALKON boasts of many other powerful features including:

- Fairing of individual curves
- Detailing of complex steel structures
- Production of drawings with augmented text (drafting automation)
- Production of data for N/C control (drafting, flame cutting, milling, etc.)
- Definition and printing of miscellaneous tabular information
- Definition of standard details (parametric descriptions of stringers, scallops, etc.)
- Formalization of design and production practice (accumulation of experience)
- Definition of identification system and data structures

The ALKON language is completely flexible, providing the user with tools to replicate the functions of literally any of the other AUTOKON modules if desired. It should be noted that while the grammar and meaning for every vocabulary word is explicitly defined within the system, the spelling associated with each of these words is not; thus each yard is able to define its own version of the vocabulary, incorporating whatever abbreviations and localized spellings are desired, taking full advantage of local customs and nuances.

One of the most powerful adjuncts of the ALKON language is represented in its ability to allow definition of norms (i.e., subroutines and functions). Norms may be given dynamic names and may be defined and called at any time. Norms are stored in the database just as any other AUTOKON information, are re-entrant, and may be executed recursively.

Since norms are, in effect, extensions of the ALKON language, the capabilities of the ALKON system are virtually limitless. In practical use within a shipyard, norms are applied in a hierarchical manner: Very basic norms are used to define elementary items such as cutouts, holes, etc. Other norms combine them into series of holes, contours with cutouts incorporated, etc. Still other norms can be used to generate complete parts to produce web frames, floors, etc. It is even theoretically possible to write a norm to design an entire ship.

The ALKON processor itself is organized in two passes, translation and processing, which are executed in sequence. All input for a run is translated before the processing pass is activated, with all data transfers between the passes being handled by the AUTOBASE routines.

Input to ALKON is normally on punched cards. This input information is read into the ALKON processor one card at a time by the Translation module. Each line is listed on the line printer and is then translated from the higher-level, external representation of words and numbers into a binary format. At the same time, a few special symbols are interpreted with appropriate actions taken, and a check is made on the formal correctness (syntax) of all the other input code. Misspelled words are rejected, missing parentheses are noted, etc. If complete fields in a statement are missing, a default value is substituted. This default feature makes it possible to write statements in a very simple way for common situations while allowing the same statements to be used in a sophisticated manner for more complex situations. Vocabulary definitions are also accommodated at this point as well as Comment statements which are recognized and immediately discarded.

All the major fixed internal logic for Translator processing is handled through the use of three sets of joblists together with vocabulary tables and decision tables. Through the use of these tables, all other statement types not mentioned above are efficiently translated from the character representation of the input to a binary (inverse Polish) notation and passed on to the Processing Module via a set of database records designated as code blocks. Newly defined norms are also retained as code blocks as are previously-defined norms.

Following completion of translation, control is then passed on to the Processor. Here the code blocks for manuscripts are returned from the database one at a time, and the statements are acted upon. If a call for a norm is encountered, the processing of the manuscript is temporarily suspended while the statements of the called code block are acted upon. If another norm call is encountered, the processing of the present code block is again suspended and the new code block acted upon. This process may be repeated to any depth, the processing of the previous code block being resumed when the processing of the called code block is completed.

When processing of the code block for a manuscript is finished, the next manuscript code block is then retrieved from the database and processed in the same manner. Finally, when there are no more manuscript code blocks, processing is completed and the present ALKON run is terminated.

The ALKON code for a simple part is shown in Figure 4. As previously indicated, the fairing subsystem results in the storage of the ship's lines in the database. The use of simple norms for retrieving this information is illustrated in Figure 5. The norm calls

FETCH TFRAME (89) AT SHELL'

instruct the system to retrieve the file containing the contour defined by the intersection of Transverse Frame 89 with the outer shell of the ship. The norm

FETCH LCON'

then instructs the system to retrieve the specific matrix in the file for the desired lofting contour. This code can then be embedded within a new contour definition, as shown in Figure 6, to define a part made up of this contour.

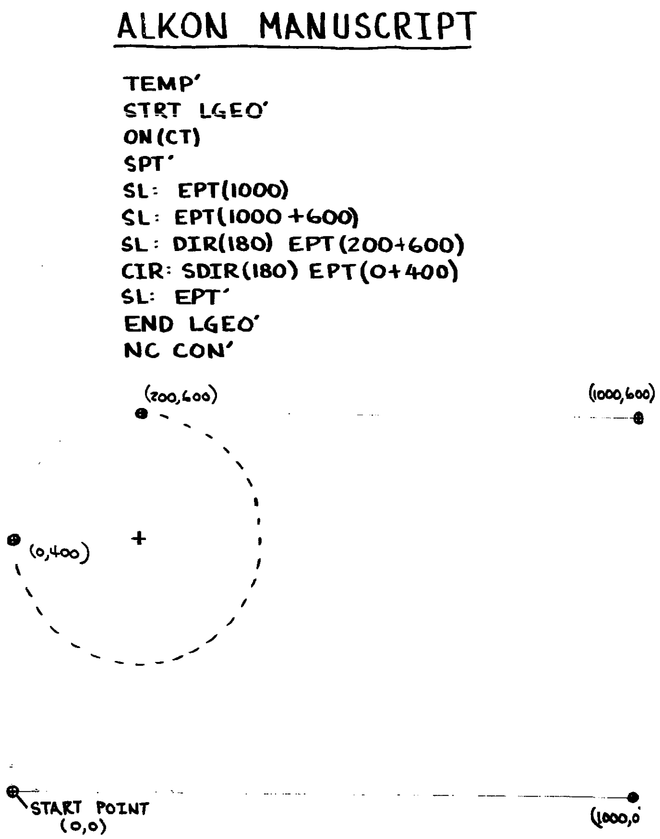
The Longitudinal Detailing subsystem generates tables of details for all required cutouts and stores them in the database. The ALKON code shown in Figure 7 illustrates how the wire frame norms can then be applied to automatically modify a lofting contour to incorporate these cutouts. The norm

GEN ACON ( )

retrieves the various tables of details, previously stored by the LANSKI module, references the appropriate norms for the cutout definitions and automatically generates the augmented contour incorporating all the desired cutouts.

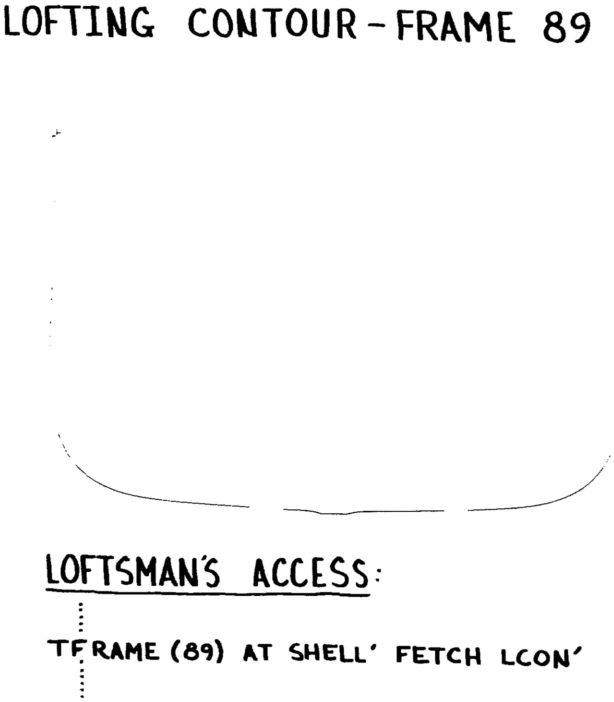
Plate development subsystem

Concurrent with the parts detailing activity but somewhat after the LANSKI runs incorporating the



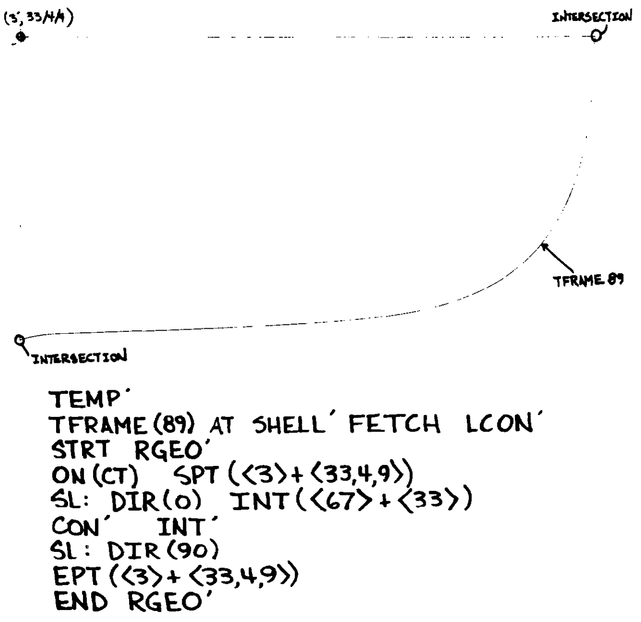
```
ALKON MANUSCRIPT

TEMP'
STRT LGEO'
ON(CT)
SPT'
SL: EPT(1000)
SL: EPT(1000+600)
SL: DIR(180) EPT(200+600)
CIR: SDIR(180) EPT(0+400)
SL: EPT'
END LGEO'
NC CON'
```

Figure 4—ALKON code for simple part



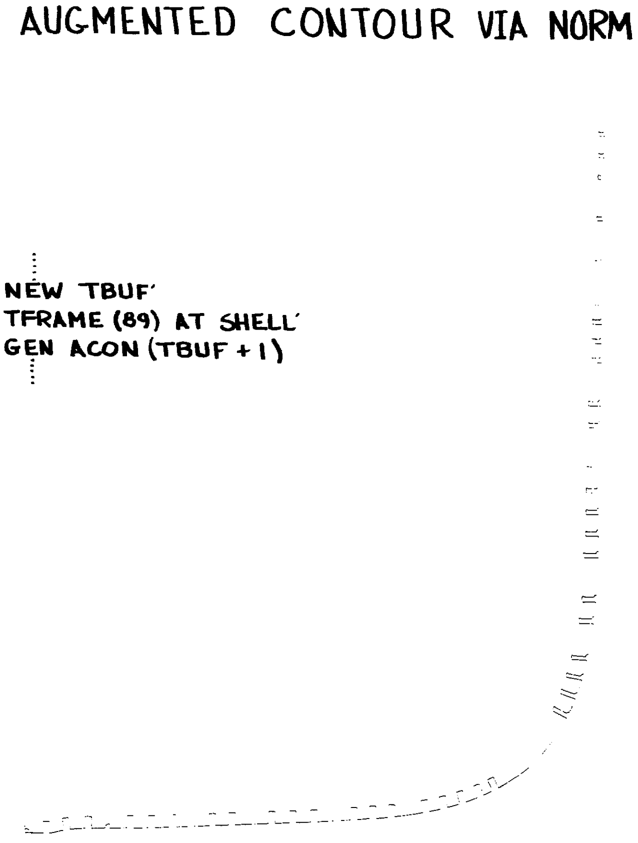
```
LOFTSMAN'S ACCESS:
  :
TFRAME (89) AT SHELL' FETCH LCON'
  :
```

Figure 5—Use of wire frame norms



```
TEMP'
TFRAME(89) AT SHELL' FETCH LCON'
STRT RGEO'
ON(CT)  SPT(<3>+<33,4,9>)
SL: DIR(0)  INT(<67>+<33>)
CON'  INT'
SL: DIR(90)
EPT(<3>+<33,4,9>)
END RGEO'
```

Figure 6—ALKON code for web frame



```
 :
NEW TBUF'
TFRAME (89) AT SHELL'
GEN ACON (TBUF+1)
 :
```

Figure 7—Augmenting lofting contour with cutouts

seams and butts in the database, use of the shell plate development subsystem normally commences. This subsystem is comprised of two AUTOKON-71 modules, aptly designated as SHELL and TEMPLATE.

Shell is used to develop the steel plate required for the outer shell of the ship and TEMPLATE generates the roll sets to be used in forming them. For each desired shell plate, input consists of butt locations (stated as axial distances from transverse frames) and seam identifications together with an indication of which of two alternate expansion methods is desired. The SHELL program retrieves the required hull form and longitudinal curve information from the database and generates 2-axis or 3-axis N/C tapes for cutting and marking the plates. It also generates N/C drawings in various scales to be used for checking, planning, or optical burning. The locations of the rolling lines and any stiffeners are automatically punch marked on the plate, or indicated in the drawing by dashed lines.

The TEMPLATE program uses essentially the same input and generates N/C tapes for plots of roll sets for the shell plates, or the roll sets themselves. Information for three templates is produced for each plate: at the first transverse frame, the last frame and the middle frame. This module may also be used to generate dimensions for the production of 1:1 wooden templates for shaping transverse frames.

Part nesting subsystem

Most parts developed with the parts programming subsystem are not directly cut but rather are stored on the database for subsequent nesting on a rectangular steel plate so that the steel is used efficiently. The AUTOKON-71 NEST module is utilized for this activity. Working with appropriately scaled part drawings, the placement of the parts on the raw plate is determined manually. Input to the NEST module then consists of the part numbers for each of the nested parts, their relative location and orientation (normal or mirror) on the plate, approximate cutting bridge placement, and (forward/backward) cutting sequence for each part. The program automatically retrieves the contour and auxiliary function information from the database for each part specified, and generates the required N/C tape for plotting and flame cutting of the nested format. The plots can be used for checking, planning or optical cutting.

Planning and production data subsystem

This subsystem is embodied in the AUTOKON-71 PRDO (**PRO**duction **DA**ta) module which is used to extract data from the database to provide useful information for planning and production purposes. Output produced includes: (1) burning time and cutting length for nested formats and single parts; and (2) area, volume, and weight information for single parts or, optionally for an aggregate series of parts.

Design analysis subsystem

This subsystem, as incorporated within the AUTOKON-71 system, is collectively designated as the PRELIKON (Norwegian mnemonic for Preliminary Lines) package. It is comprised of 23 modules which are used for both preliminary and final naval architecture calculations and for automatic production of various tables and plots normally required on delivery of the ship.

All computations are performed on an independent database. Initial establishment of this database is achieved with the Hull Definition module or the Hull Variation module, the latter of which can be used to apply minor changes (LCB,CB, or main dimensions) to elements of a library of hull forms.

The AUTOKON-PRELIKON linkage module, which, in effect, retrieves the body plan from the AUTOKON database and maps it into the PRELIKON database, may be used later to allow the modules below to be applied for final calculations.

Modules are also provided for the following computations:

- Hydrostatics
- Load Distribution and Balancing
- Resistance
- Bon Jean Tables and Plots
- Transverse Stability
- Floodable Lengths
- Launching
- Trim Tables and Plots
- Capacity, Sounding, and Ullage Tables
- Grain Stability
- Longitudinal Strength

Program support activities

The AUTOKON Support Program can be defined as a special interest activity being pursued under the auspices of the REAPS Program. As previously stated, the basic purpose for this program is to provide an organized mechanism for maintaining and enhancing the AUTOKON-71 system for optimal use in the U.S. shipbuilding environment. Activity towards this end falls into three categories: Software Support, Information Dissemination, and System Enhancements Requirements and Planning.

For software support, all the expected system support activities are being pursued under this effort. A formal mechanism has been established for reporting system failure information for subsequent resolution by the REAPS Program staff. Periodically, all the accepted fixes are incorporated into the system and a new Standard System is distributed to the participating yards. Tapes and complete update documentation are provided. Implementation assistance is also made available to new system users. A new four-volume AUTOKON Users Manual has been written and made available to participating yards, and training courses have been developed and given.

Proposals for major system capability enhancements are periodically reviewed by the AUTOKON Support Program Technical Representatives, with resultant recommendations made for MarAd funding for those deemed to merit implementation.

PIPE DETAILING SYSTEM

The following describes the results of a preliminary systems design recently carried out by Newport News Shipbuilding and Dry Dock Co. in the first phase of a two-phase effort to design and implement a minimum cost system for pipe detailing. The purpose of this effort was to identify an input station configuration for use in digitizing piping design data that would substantially reduce the cost of preparing input data for various computer-based systems used for the production of pipe manufacturing documents.

Operational overview

Functionally, the desired system must accommodate four major tasks: data input, document production, data revision and updating, and data communication. These tasks are described below in the context of a typical operational sequence.

At the beginning of a typical design session, the designer will retrieve the appropriate piping arrangement plan or preliminary design drawing and mount this document on an X-Y digitizer, along with the command menu list to be used in the input process. Next he must retrieve from a hull structural database (e.g., an AUTOKON database) the necessary structural reference data (transverse frame, deck, bulkhead locations, etc.) such that geometric data for the piping system may be conveniently referenced to these locations. Using these reference locations, the designer will then calibrate the drawing(s) on the digitizer by touching the positions of these structural landmarks with the free cursor. The system is then prepared to accept digitizer input.

The designer then proceeds to input the piping system geometry and details utilizing the digitizer, a keyboard or CRT cursor picks. These picks relate to a CRT display of the current definition of the piping system which is built up during the input process. The CRT display also serves as a check on the designer's work as he proceeds.

Keyboard input of piping geometry may be in the form of coordinates relative to the structural reference locations retrieved from the hull database, absolute coordinates or coordinates relative to other points already defined in the pipe geometry itself.

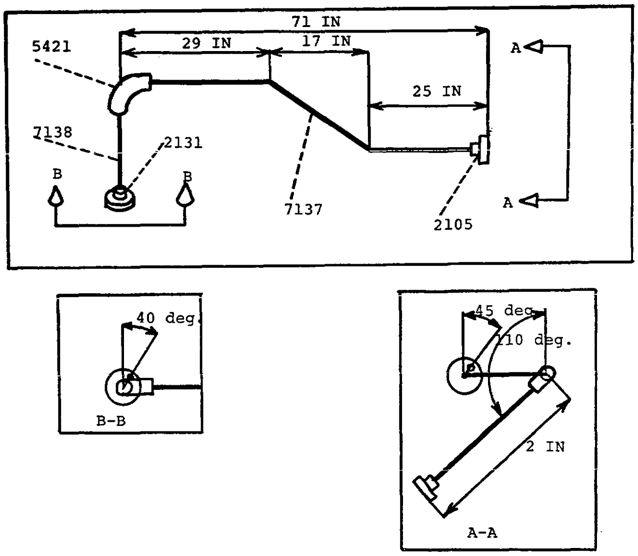
An important feature of the system is the capability to automatically select specific components based on a predefined set of component selection rules applicable to a particular piping system. This feature, for example, will allow the designer to specify a condition generically (e.g., TURN) during the input process. Then, a post processing operation will automatically determine the specific fitting, etc., to be substituted (e.g., a bend or a 90° or 45° elbow) for the generic condition through an analysis of the component selection rules.

Regardless of whether components are explicitly referenced by the designer, or automatically selected as described in the preceding, the system will proceed to automatically determine the required orientation in space of the component based on its local geometry and the geometry of the connected piping. Additionally, error checks are performed on the geometry of bent pipes, to insure that they may be produced by the yard's pipe bending machines, as well as on joints in the piping system to verify the compatibility of members at the joint in terms of diameter, end type, and alignment.

At the completion of an input session, or at any time during the input process, the designer may request the system to produce checking documents. These include a centerline check print (plot), consisting of a single line drawing of piping centerlines drawn to the same scale as the drawing from which the piping geometry was lifted such that it may be overlaid on the original, a piping system data list, and a listing of missing data (see Figure 8). This latter list is required if the designer is working from a preliminary design drawing which will not contain all necessary detail information required to complete the design. In this case, the system notes the locations where information is lacking, but allows the designer to proceed with the input process. Then, at a later date, when the designer has determined which details are to be used for this specific system, he can return to the incomplete system definition to fill in the missing information.

Figure 8 is an example of such a checking document as might be produced by the system. Note that graphical prototypes of the defined fittings are also presented. Also indicated are two sectional views of the system, defined by the user. The user may select views of the system by one of three methods:

1. by identifying several structural reference points which are to be included in a plan, section or elevation view;
2. by identifying a pipe leg which will serve as a view axis or a pair of intersecting legs the normal to which will serve as the view axis;
3. by specifying cross-sections through an existing view (e.g., Figure 8).



ID	TYPE	SIZE	SOURCE
2131	Flange	3 in	SN71344
2105	Flange	3 in	PO591-73-661
5421	Elbow	3 in x 3 in	PO591-77-005
7137	Pipe	3 in x 62.5 in	SN62131
7138	Pipe	3 in x 12.0 in	SN62131

Figure 8—Typical pipe detailing system plotter output

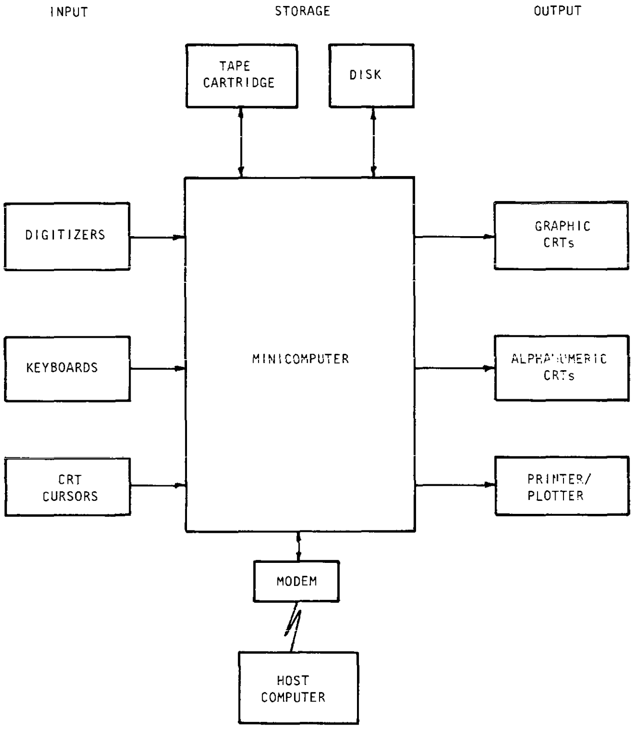


Figure 9—Hardware environment for pipe detailing system

At the conclusion of an input or modification session, the designer may either dump the specific piping system database to magnetic tape for later retrieval and additional detailing work or, if the system is complete, transmit the database to the central database for accessing by engineering analysis, material and production control, and manufacturing document production programs.

System configuration

Figure 9 is a schematic representation of the proposed input terminal configuration. Each such terminal may support from one to four input stations each consisting of a digitizer, a keyboard, an output graphical display device and an output alphanumeric display device (either a teleprinter or character CRT). The box labeled "Printer" may actually consist of one of two alternative configurations; an incremental pen plotter for the production of centerline check prints and a medium speed line or character printer for producing data lists, or a single electrostatic printer/plotter which would satisfy the requirements for both applications. The communications interface and modem will support 1200 baud synchronous communications between the minicomputer and the central site facility, a Honeywell 6080, via a Datanet 355 communications front-end.

All database operations will be implemented within the framework of a vendor-supplied database management program which conforms to the recommendations of the CODASYL committee. A database network structure has been defined for the representation of piping data interrelationships which uses a subset of the CODASYL specifications and which is compatible with several commercially available minicomputer-based DBMS.

The final, or development phase of the work reported here is currently under way at Newport News and is scheduled for completion in early 1978.

SHIPYARD GRAPHICS AND COMMUNICATIONS TERMINAL

The following discussion reports on the results of a design effort carried out by IIT Research Institute to develop a low cost remote terminal configuration for use by shipyard loft departments in processing the input and output of N/C software systems such as AUTOKON and others.

Background

This project was initiated with the intent of developing a terminal design which was modular, for ease in configuring various throughput/cost configuration alternatives, as well as to minimize the cost of each such configuration throughout the range of these capability alternatives.

Functional requirements

The functional requirements of the terminal, developed as a result of interviews with the personnel of various shipyards in the U.S., are as follows:

1. The terminal must provide a Remote Job Entry (RJE) capability for a variety of mainframes via emulation of various remote batch terminal protocols typically associated with these mainframes.
2. It must provide a local facility for reviewing numerical control (N/C) data graphically on a cathode ray tube (CRT) display and on an automatic drafting machine (ADM) in the following modes:
   a) ESSI verification
   b) Drafting

   Additionally, the system must be capable of punching, on paper tape, the final verified ESSI file for transmittal to the burning shop to be used to direct N/C flame cutters.
3. It must provide the necessary data management and storage facilities to support the above-mentioned activities efficiently.
4. It must finally, provide an operating environment in which software tasks associated with the above functions may execute concurrently in servicing one or more users.

Operational environment

The terminal user/operator, as with other remote batch operations, will load the appropriate RJE emulation software into memory from local mass storage and establish communications with the central site facility. Once loaded, the emulator is prepared to accept for transmission a user input deck. The user loads the deck in the card reader and instructs the emulator to transmit the input stream to the central site.

At the central site, the appropriate N/C software system module is invoked and executed and the output of the run queued for transmission back to the originating terminal site. Output from these programs is of two forms: the normal printed output (or Print File) which will be directed to the terminal's line printer, and the resultant ESSI code describing the geometry of the part, plan, etc., produced by the run. This file will be treated by the central site as a Punch File and addressed to the terminals' punching device. In practice, however, this file will be automatically written to the terminal's local mass storage device and catalogued under a file name consisting of the unique run identification information assigned by the central site.

Upon reception of these files, the user may then referring to the run I.D. appearing on this printed output, initiate a graphics session at the CRT console and call out the appropriate "Punch File" for graphics processing.

The graphics processing system will allow the user to plot the ESSI data on the CRT for a quick check of the data or produce an ADM drawing of the data by simply specifying the desired output device. If the user of the CRT is not the originator of the run, this terminal operator may simply produce a hardcopy of the CRT check plot on the optional hardcopy device for delivery, along with the printed output listing, to the run's originator for review. After review of the run results, the loftsman or designer who initiated the run may request the operator to either plot the catalogued Punch File on the ADM, if the results are satisfactory, or purge the file if an additional run will be required. A punch tape of the file will also be requested if the file is to be used by the N/C burning shop.

System description

Figure 10 represents a typical configuration of the terminal being described. One of the more interesting features of the system is the block labeled "Table Control." This is a microprocessor-based control unit containing all the required logic for linear and circular arc interpolation and the generation of slope and acceleration control required by the head drive electronics of the ADM itself. This capability relieves the system processor of the requirement of computing and outputting incremental positioning data for each step of the ADM's resolution (i.e., 0.001").

The controller can be interfaced to the system processor via standard EIA serial interfacing, further simplifying the overall terminal configuration.

The CRT device is of the storage tube variety and is also interfaced to the system via an asynchronous EIA RS 232 interface. It will operate nominally at 9600 baud. It is equipped with a thumbwheel cursor for use in the graphics processing software for digital coordinate input.

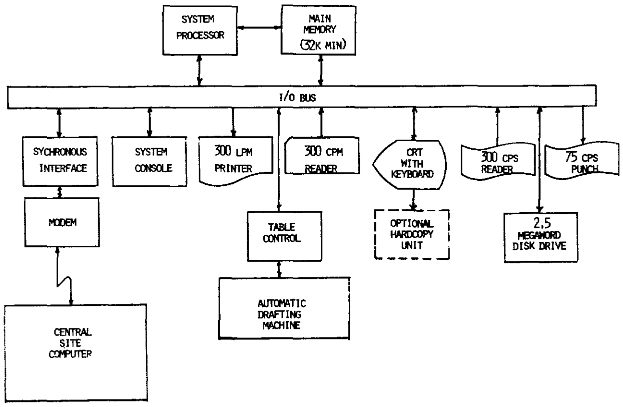


Figure 10—Typical RJE/graphics terminal configuration

The system console may alternatively be a character CRT display or teleprinter device.

Graphics software

As mentioned previously, two distinct modes of operation of the graphics processing system are available. The first, or ESSI verification, mode of operation in addition to producing plots of geometry described by the ESSI data, also checks the ESSI auxiliary function codes (e.g., center torch on/off, rapid traverse on/off, left/right kerf width compensation on/off, etc.) for consistency. In this mode, auxiliary function code annotation is added to the geometry display/plot and errors reported (e.g., both center torch and rapid traverse on concurrently) on the user console.

The second, or drafting mode, ignores all but the center torch auxiliary functions which serve as the pen up/down commands. This mode is of particular value in reviewing ESSI output from the N/C software modules which are not directly concerned with flame cutting. These would include, for example, the AUTOKON FAIR and LANSKI (see the discussion of AUTOKON in this paper) modules which output ESSI representations of hull lines (e.g., bodyplan, buttocks, waterlines, shell plate seams, etc.) and longitudinal structure traces. For these applications, the user may make use of the graphics processor's windowing (or zoom) capabilities to good advantage to examine more closely potential problem areas requiring greater visual definition. Also implemented in the graphics system software are 3-dimensional display and transformation routines to support future applications in the design area.

Capabilities summary

A modular, multi-user RJE/Graphics terminal system for shipyard use has been designed. The system's capabilities (summarized in Figure 11) include RJE communications, for a variety of main-frames, which may be executed concurrently with one or more graphics processing sessions or with various utility programs operating in background mode. The graphics processing software allows interactive view modification at a CRT or a simple batch type mode of operation applicable to both CRT and ADM.

N/C FRAME BENDING MACHINE

The following describes a development effort carried out by Case Western Reserve University which has resulted in the production of a prototype, fully automated frame bending machine.

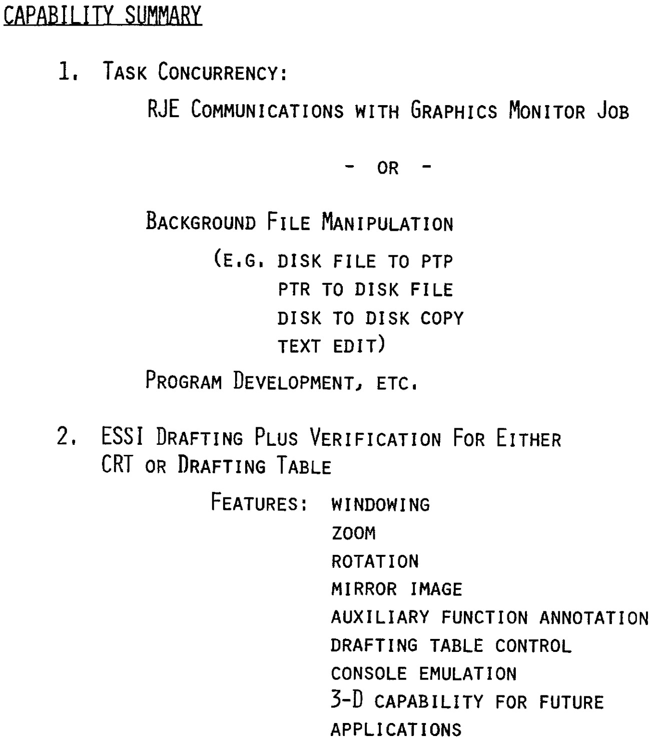
TABLE I—FRAME BENDER CONTROL ALGORITHM

1. OPERATOR INSERTS BEAM INTO FRAME BENDER
2. OPERATOR CLAMPS TRANSDUCER AT END OF BEAM
3. INPUT THE DESIRED BEAM SHAPE
   A. ESSI model of circular arcs and straight line segments (can be piecewise linear)
   B. From AUTOKON paper tape, teletypewriter, or disk file
   C. ASCII or EIA character codes
4. INPUT THE BEND PARAMETERS
   A. Work length minimum and maximum
   B. Initial unbent length at end of beam
   C. Clamping mode
   D. Tolerances for feed distances and bend angles
5. CALIBRATE TRANSDUCERS WITH A/D CONVERTERS
6. SET UP IDEAL MATHEMATICAL MODEL OF BEAM
   A. Work points are preferred at junctions of ESSI elements
   B. The last work section may overlap the second last one
   C. The tangent vector at each work point is determined from the ESSI model
   D. A table summarizing ideal model of beam can be printed
7. EACH WORK SECTION IS PROCESSED AS FOLLOWS:
   A. Feed beam and adjust moving head to position new work section between the fixed and moving heads
   B. Update model of actual beam to reflect feeding
   C. If necessary, move transducer to new point on beam, find new reference point on model of actual beam, and find corresponding point on ideal model
   D. Calculate (X, Y) aim coordinates for transducer reference point from ideal model
   E. Bend to Y coordinate of aim point, release, and measure springback
   F. Until bend angle tolerance is satisfied (but never more than 2 iterations), recalculate required "overbend" based on springback just observed, bend to new Y coordinate, release, and measure springback
   G. Update model of actual beam to reflect bending of this work section
   H. Feed to X coordinate of original aim point
8. WHEN ENTIRE BEAM IS FINISHED, OPTIONALLY PRINT A TABLE SUMMARIZING THE MODEL OF THE ACTUAL BEAM
9. OPTIONALLY PRINT A TABLE COMPARING THE IDEAL MODEL WITH THE MODEL OF THE ACTUAL BEAM

Background

Historically, frame bending in the shipyard has been carried out by one of two methods: hot slabbing, which requires the shape to be furnaced until the material becomes plastic and then forced against a full-sized template through use of a hydraulic ram, or three-point cold form bending, in which the operator of the three-point machine iteratively applies a force on the shape between two fixed support points until the desired curvature has been produced.

Each of these techniques has substantial drawbacks. Both require the production of full-sized templates. The hot slabbing process requires a furnace facility



CAPABILITY SUMMARY

1. TASK CONCURRENCY:

   RJE COMMUNICATIONS WITH GRAPHICS MONITOR JOB

   - OR -

   BACKGROUND FILE MANIPULATION

   (E.G. DISK FILE TO PTP
   PTR TO DISK FILE
   DISK TO DISK COPY
   TEXT EDIT)

   PROGRAM DEVELOPMENT, ETC.

2. ESSI DRAFTING PLUS VERIFICATION FOR EITHER CRT OR DRAFTING TABLE

   FEATURES: WINDOWING
   ZOOM
   ROTATION
   MIRROR IMAGE
   AUXILIARY FUNCTION ANNOTATION
   DRAFTING TABLE CONTROL
   CONSOLE EMULATION
   3-D CAPABILITY FOR FUTURE
   APPLICATIONS

3. EXPANDABILITY

   TO 128K WORDS MEMORY
   TO 8 DISK DRIVES
   MULTIUSER
   MULTITABLE

Figure 11—Shipyard graphics terminal capabilities summary

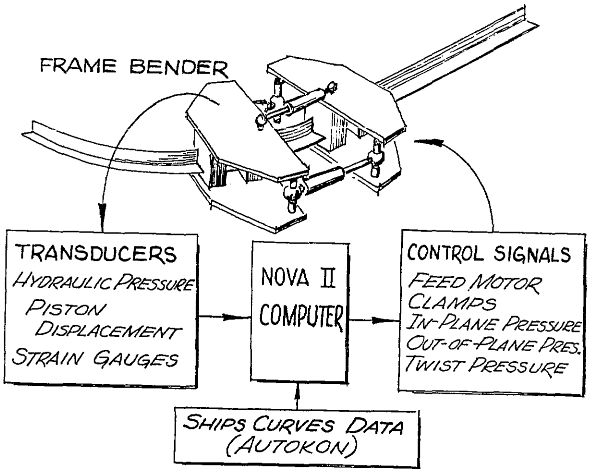


Figure 12—Control loop for automatic frame bender

and is very labor intensive. And the cold bending is slow, requires a skilled machine operator and, most significantly, produces out-of-plane bending, buckling and twisting (i.e., the plane of deformation of the shape and the plane in which the bending moment is applied may not coincide).

The CWRU frame bender represents a unique solution to these problems. The prototype itself is a 1/6 scale version of a full scale device which can accommodate interchangeable dies, automatic clamping, automatic feeding and provision for operation in either a completely autonomous, computer-controlled mode or by manual control.

Mode of operation

Figure 12, reproduced from Reference 2, depicts the control loop involved in directing the machine. Feedback signals from the transducer group are fed through a multiplexed 12-bit A/D converter to the computer (a 32k Nova 2/10). Based on the current and desired frame contour (acquired from the ship structural database, e.g., AUTOKON) the computer calculates the necessary bending to be applied and outputs this data, via a D/A converter, to control hydraulic valves on the machine. This process is an iterative one in which short segments of the frame, called work sections, are processed sequentially. For each work section a target or "aim" point is computed such that cumulative error is minimized. The work section is then bent, taking into consideration the approximate springback of the frame, in an attempt to position a reference point near the original end of the frame on the aim point. If the two points do not coincide after bending and springback, the work section is again bent on the basis of the springback measured after the last bending operation and the deviation of the reference point from the aim point. Throughout this process, for frames with non-symmetrical cross-section, hard wired logic maintains the appropriate in-plane bending through control of a servo valve.

Three separate mathematical representations of the frame are maintained by the minicomputer during the bending process. The first is the ESSI (i.e., N/C contour) model of the frame as retrieved from the AUTOKON database. The "ideal" model is derived from this ESSI model prior to the initiation of bending by segmenting the ESSI elements into work sections (see Figure 13) according to a set of segmentation rules, and computing vectors tangent to the ESSI model data at all work points for alignment of the frame. Finally, the actual model, developed during the bending process, represents the true shape of the frame based on actual curvatures applied and work section lengths fed. From this model, a table of discrepancies can be produced relating the final actual to desired shape.

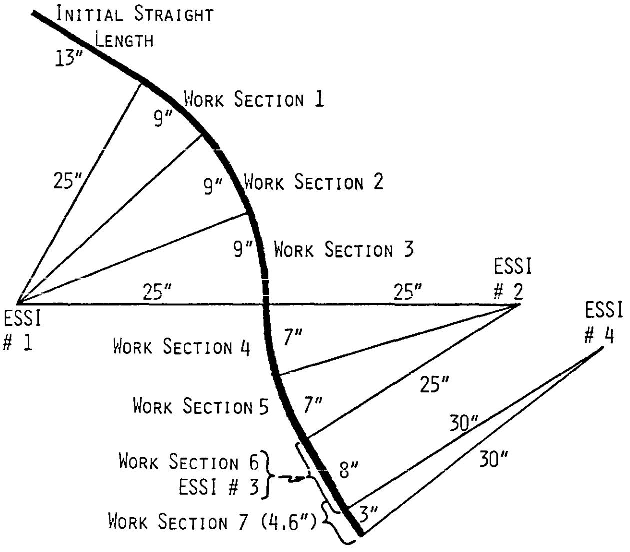


Table of ESSI Elements (100 units/inch)

1322	-2205	-1178	-2205	-
382	-1328	2500	0	+
425	-678			
172	-246	2542	1594	+

Figure 13—Example of an ESSI model and its ideal work section model

FUTURE PROGRAMS

Consistent with a broadening of the technical scope of, and shipyard participation in the REAPS Program, is a shift in emphasis in the program's R&D efforts to the development of programs or systems which do not rely on any of the several individual N/C software or other production-oriented systems for applicability. Some examples of projects to be undertaken in the near future are listed below.

1. Structural Detailing System—for definition of stiffener intersection details including N/C descriptions of end-cuts for subsequent use by N/C fabrication equipment.
2. A Low Cost Parts Definition System—for quickly entering or modifying existing part geometries in an interactive mode through use of a minicomputer-based digitizer system.
3. Sheet Metal Template Generation System—for use in developing sheet metal ducting details and subsequent template preparation.
4. Structural Assembly Aids System—for producing parts explosion type drawings of structural units to assist assembly crews in fabricating structural units quickly and accurately.

Such developments it is felt will now and in the future provide U.S. shipbuilders with cost-effective technological problem solutions which will serve to enhance shipbuilding productivity.

REFERENCES

1. "AUTOKON Users Manual," Vols. 1 to 4, IIT Research Institute, 1975, 1976.
2. Braun, D. C., "The Case Western Reserve N/C Frame Bending Machine," *Proceedings of the 1975 REAPS Technical Symposium*, June 1975, pp. 219-246.
3. Landmark, Anton, "ALKON—A Language for Algorithmic Design," *ICCAS (International Conference on Computer Applications in the Automation of Shipyard Operation and Ship Design) Proceedings*, August 1973.
4. Martin, D. J., "A Graphics and Communications Terminal for Shipyard Applications," *Proceedings of the Thirteenth Annual Numerical Control Society Technical Conference*, March 1976.
5. Moore, R., "The AUTOKON System at Newport News," *Proceedings of the 1974 REAPS Technical Symposium*, June 1974, pp. 39-50.
6. Rourke, P. W., "Software Engineering for Digitizer/Minicomputer-Based Piping Data System," *Proceedings of the 1975 REAPS Technical Symposium*, June 1975, pp. 89-110.

Computer analysis and evaluation of marine structures

by DONALD LIU and MATIAS E. WOJNAROWSKI
American Bureau of Shipping
New York, New York

ABSTRACT

The marine industry in general and the American Bureau of Shipping in particular have turned to the extensive use of computers for the solution of the problems encountered in the design, analysis and construction of ships and other marine structures. Mathematical solution techniques have been available for some time, but the complexity of the necessary calculations precluded their use. The advent of electronic digital computers with their powerful, constantly expanding "number-crunching" capabilities has bridged this gap, leading to the application and further expansion of available techniques.

The most useful and usable tool is the finite element method, which has found widespread use and has been extensively applied for structural, dynamic and thermal analyses of marine structures, ranging from entire vessels to local structural details. Descriptions and examples of such analyses are given in the paper.

Other facets of computer versatility applicable to and used by the industry include conversational programs, naval architecture programs, computer graphics and information retrieval. These subjects are presented and discussed in the paper.

Looking ahead, future trends of computer applications in the industry are mentioned.

INTRODUCTION

The American Bureau of Shipping (ABS) is a ship classification society which performs the primary service of certifying the soundness and seaworthiness of merchant ships and other marine engineering structures. ABS establishes standards known as "Rules"[1] for the design, construction, and periodic survey of vessels. By applying these internationally accepted "Rules", ABS classes ships, that is, assures that ships are fit for their intended service. Classification provides assurance to owners, purchasers, shippers, underwriters, and other interested parties that a particular vessel possesses the structural and mechanical capability for safe performance.

Realizing the power of the computer and its application as a valuable engineering tool for vessel classification services and such fields as naval architecture and structural engineering, the American Bureau of Shipping has been in the forefront of the development and usage of computer programs in the marine industry. As a service organization to the industry, ABS has also adopted a policy of making known to interested parties the methods it employs in computer-related activities. This paper is a small contribution toward that goal.

The complexity of engineering problems encountered in the marine industry has led to extensive and ever-expanding computer usage.[2] In addition to the conventional static and dynamic problems of design, the problems encountered in assessing marine structural response are compounded by the unpredictable nature of sea, and sometimes cargo, loads. Sea conditions represented by wave heights, wave lengths and wind speeds are measured and the information is compiled statistically. Structural design is based on sea conditions with a probability of occurrence of 10^{-8}, which corresponds to an event occurring once during an assumed ship service life of approximately 25 years (20 years at sea). Special phenomena associated with the dynamic interaction of waves and ships at sea must be taken into account. Springing, for example, is a vibration of the complete vessel induced by the wave frequency in conjunction with the ship's elastic properties. Other areas of concern are local vibrations, which may be induced by waves or action of the propeller and drive shafts. Other loading conditions include those due to thermal effects, sloshing of liquid in cargo tanks and sea ice.

Mathematical techniques, such as matrix methods, finite element methods and statistics, have been available for a long time. It has taken the advent of electronic digital computers to make possible the full utilization and implementation of these techniques into an efficient engineering approach to the solution of the numerous problems associated with the design, construction and analysis of ships and other marine structures.

FINITE ELEMENT METHODS

The finite element technique is a relatively new and very useful method for stress analysis of structural continua. The method relies strongly on the matrix formulation of structural analysis introduced mainly as a result of the increasing use of computers. There has been a concurrent and rapid development of electronic computers, matrix methods and finite element techniques.

In the finite element method, a solid continuum is subdivided into an assemblage of discrete elements of finite dimensions.[3] In effect, the real system or structure is modeled by a simplified, idealized system for which a solution is available. The idealized model is then analyzed by one or more of the available methods of analysis.

The finite element technique has a sound theoretical basis within the framework of the classical theory. It may be interpreted as a close relative to the well-known Ritz method, in which the displacement field in a continuum is usually described by means of a sum of preselected functions, each multiplied by a constant. The constants are determined by means of the condition of minimum potential energy.

While in the classical Ritz procedure one set of functions describes the displacement field in the entire continuum, the finite element method assumes individual displacement fields for each of the elements. The internal displacements in the elements are uniquely defined in terms of the nodal point values, and the entire displacement field is assumed to consist of a large number of piecewise continuous fields, each covering the extent of one element. The conditions of equilibrium of the nodes may be shown to yield the displacement field corresponding to minimum potential energy for the selected displacement pattern. As in the Ritz procedure, the solution will generally be approximate, but the method converges toward the correct solution.

Many new developments and refinements are constantly being added to the existing finite element programs. One of the most promising of these extensions, extremely useful in the analysis of complex structures, is the substructuring capability. Substructures are assemblages of basic elements (beams, plates, etc.) which serve as building blocks for the representation of larger, more complex structures. Substructuring decreases the amount of input data required to generate the structure, particularly for repetitive structural arrangements. It also decreases the number of unknowns encountered in the solution of the problem, with a corresponding reduction in computer requirements.

An improved substructuring concept involves the use of the "reduced substructure" technique,[4] in which kinematic constraints are applied on the boundary so as to reduce the number of interaction freedoms. This insures displacement compatibility along the boundaries between adjacent substructures and/or elements.

Some examples of the numerous finite element applications in the marine industry are given in the following sections.

STRUCTURAL ANALYSIS

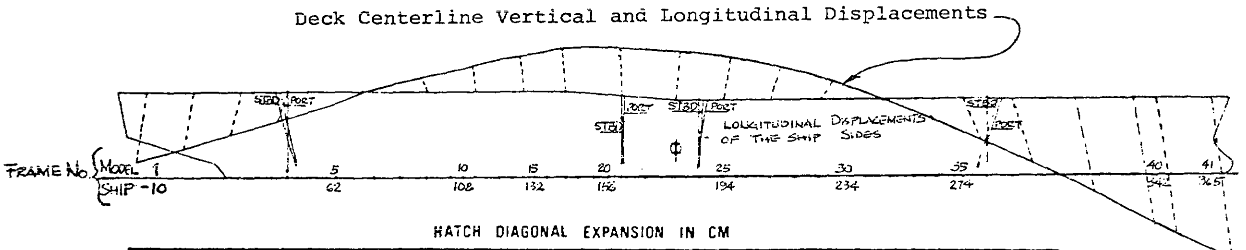
Discretization is an essential part of the structural analysis problem.[5] In general, discretization introduces approximations into the analysis; for example, a finite element idealization generally involves approximations both of the geometric form of the structure and of the displacements which it develops. The degree of refinement which is employed in the finite element mesh should be based on the analysis requirements, i.e., a fine mesh is needed in regions where the structural behavior is most complex. Moreover, the mesh must be particularly fine if the primary objective of the analysis is stress distribution rather than deflection, because the derivatives of the displacements are represented less accurately than are the displacements themselves when applying the more commonly used displacement method.

The finite element method has been extensively applied to all types of marine structures. The first applications were for the analysis of supertankers (large tankers with deadweight tonnage above 200,000 dwt), as a result of their rapidly increasing size and associated changes in ship configuration.[2] Classification society rules based on previous design and experience were not adequate for these new ships, and considerable structural damage was sustained by some of the earlier supertankers. Computerized structural analyses have provided the means to overcome Rule limitations and to provide the required structural integrity and design efficiency.

Other marine structures such as containerships, ore or bulk carriers, liquefied natural gas (LNG) carriers, submarine structures, surface effect ships, hydrofoils, ice breakers, drilling rigs (fixed and floating), offshore platforms, etc., have been extensively analyzed by computerized methods.

Some examples of these analyses by means of the ABS/DAISY (**D**isplacement **A**utomated **I**ntegrated **SY**stem) program[6] are shown in Figures 1 through 4.

Figure 1 shows the displacements of the deck and the lateral expansion of the front hatches of a container vessel in oblique waves, subject to combined vertical, lateral and torsional loads.[7] Figure 2 shows the extent of a three-dimensional model of an LNG carrier.[8] Figure 3 shows a typical transverse web frame of an oil tanker, its idealization by finite elements, calculated displacements and corresponding stress contours. Figure 4 shows the various design improvements of a shell longitudinal connection to a horizontal oiltight bulkhead girder in a supertanker.[9]



HATCH DIAGONAL EXPANSION IN CM

HATCH NO.	1	2	3	4	5	6	7	8	9
d	2048.197	2046.749	2494.283	2494.189	2969.781	2969.800	2965.790	3036.431	3036.469
EXPANSION	0.382	0.634	0.896	.125	1.510	2.056	1.989	2.413	1.695

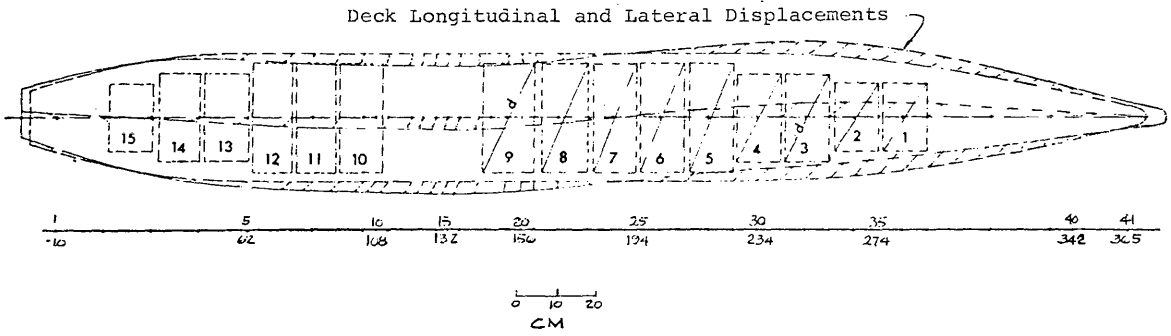


Figure 1—Overall displacements of deck and front hatches diagonal expansions

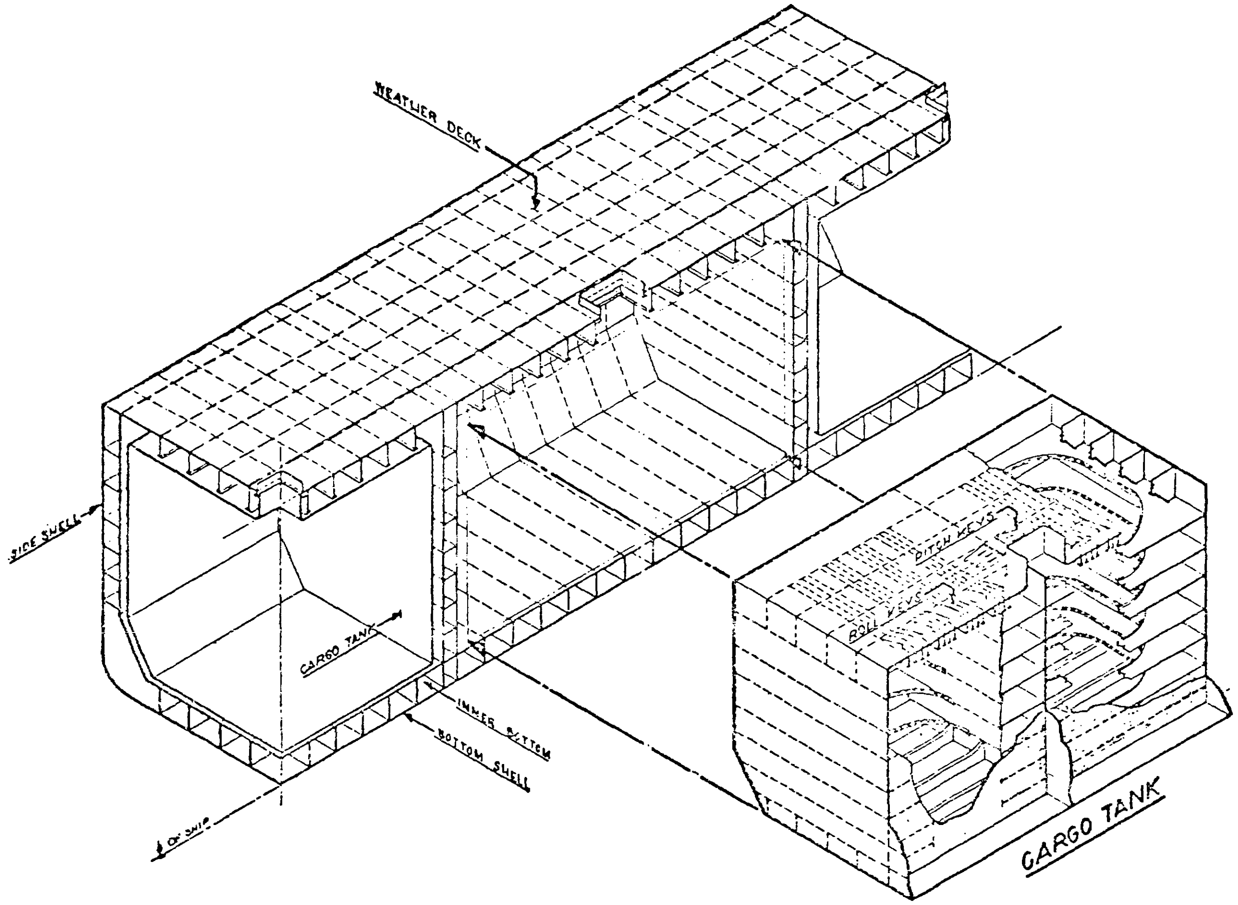


Figure 2—Extent of three-dimensional model for independent tank LNG carrier

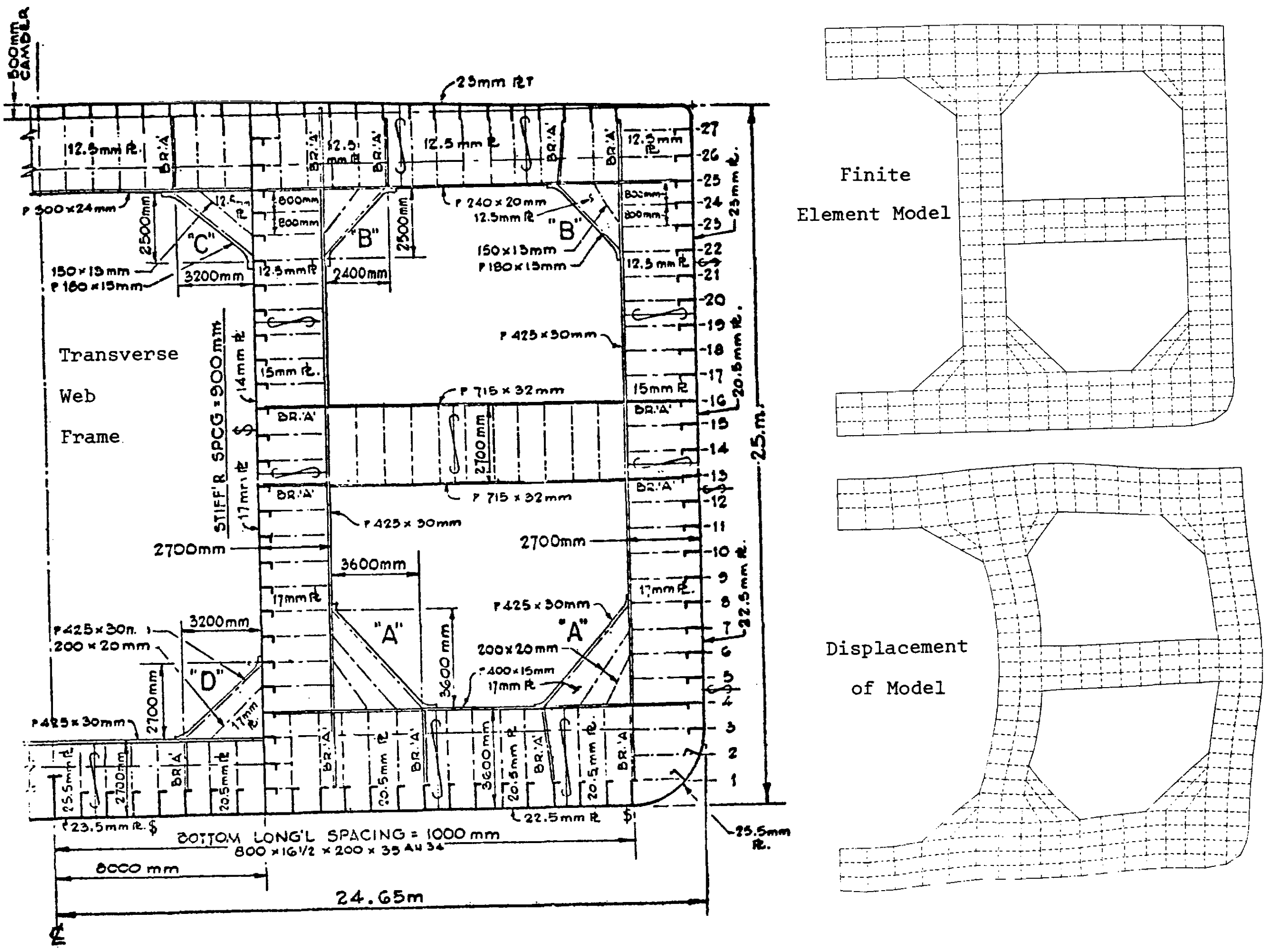


Figure 3—Two-dimensional web frame analysis

STRUCTURAL DYNAMICS

Generally speaking, the concepts of structural dynamics are not new; both mode-superposition procedures and direct time-integration analyses of the equations of motion were well understood many decades ago.[5] However, the finite element method provides a unified approach to discretization which can be applied to completely arbitrary and highly complex structures, and the modern digital computer makes possible the routine solution of the resulting equations of motion, which may involve hundreds or thousands of degrees of freedom evaluated at hundreds of time intervals during the dynamic response. Thus, these new tools make it possible to meet current structural dynamic analysis needs in fields such as the design of offshore oil drilling platforms for wave, wind and earthquake loads, the design of supertankers and other ships for wave loads and other hydrodynamic forces, etc.

Some examples of the application of computerized dynamic analysis are shown in Figures 5 and 6.

Figure 5 shows the idealization of an entire cargo ship for the study of vibrations and slam response by means of the SHVRS (Ship-Hull-Vibration Response System) program.[10] Figure 6 shows an isometric view of the model used for the vibration analysis of a supertanker.[11] The model consists of one-half of the entire vessel, and was plotted with the aid of the SAP IV (Structural Analysis Program) plotting routines.[12] Figure 7 shows the wave-induced, springing and combined bending stresses at the midship section of a Great Lakes ore carrier as a function of the wave frequency. This was calculated and plotted with the aid of dynamic analysis programs being developed at the American Bureau of Shipping.

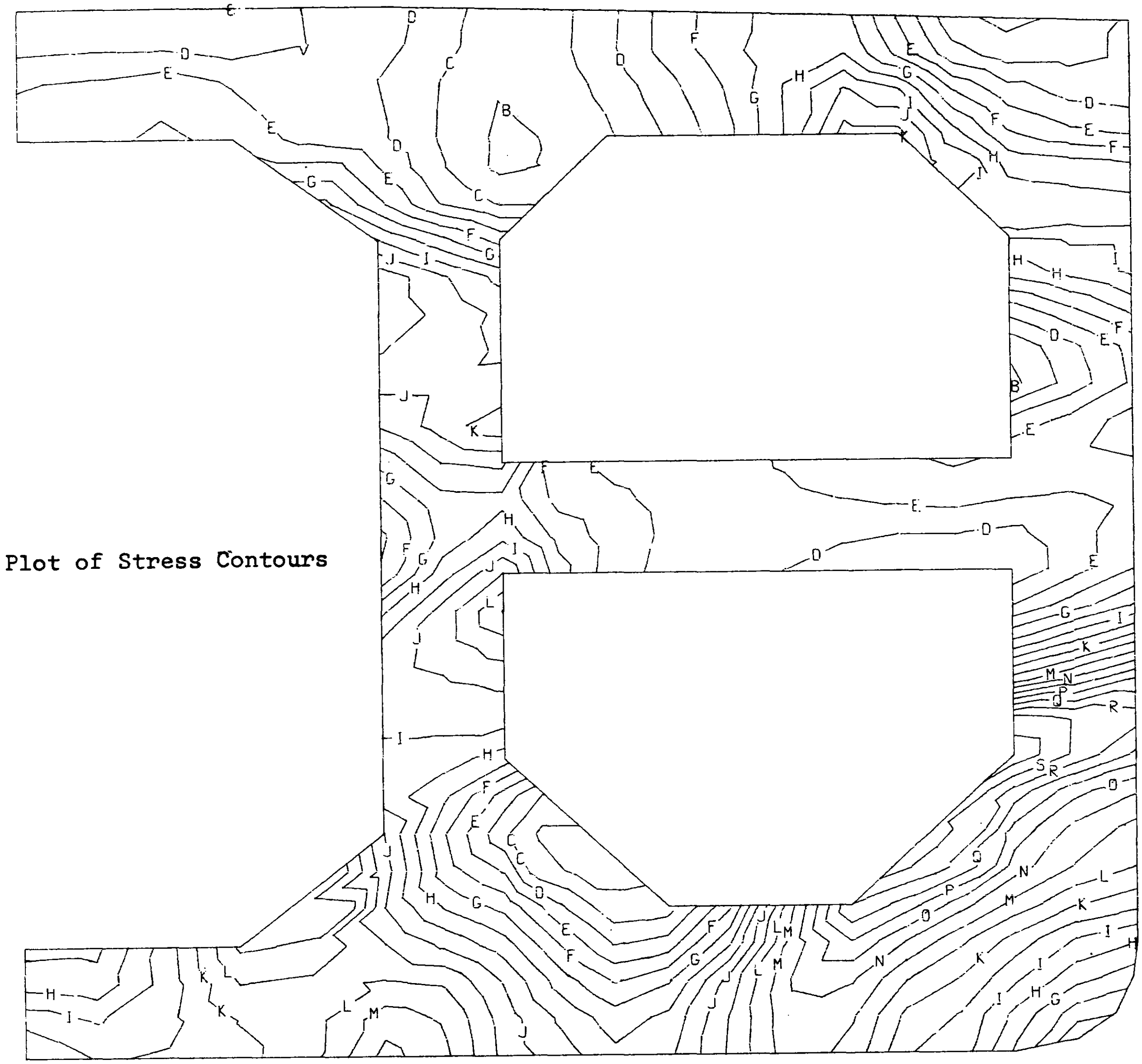


Figure 3 (continued)

THERMAL ANALYSIS

Thermal stresses and deformations are produced by changes in temperature distribution, and high temperatures influence the mechanical properties of a material to a great extent.[13] Ship structural members and machinery parts are often subjected to various kinds of heat treatment, producing thermal stresses which may result in residual deformation. Temperature differentials of cargo and surroundings, as for example in the high temperature range of hot asphalt (300° F.) or in the very low temperature range of LNG (−260° F.) produce high thermal stresses and deformations which must be considered in the overall design and analysis of a vessel. The magnitude of thermal stresses developed in a hull structure is governed by the restraints provided by surrounding structure and by the non-uniform temperature differences in the hull due to the ship's internal and external environment.

The finite element method is a powerful tool to obtain thermal stress distributions and deformation patterns. Some examples of its application are shown in Figures 8 and 9. Figure 8 shows thermal loads and calculated stresses on a typical transverse web frame of an asphalt tanker carrying cargo at temperatures of up to

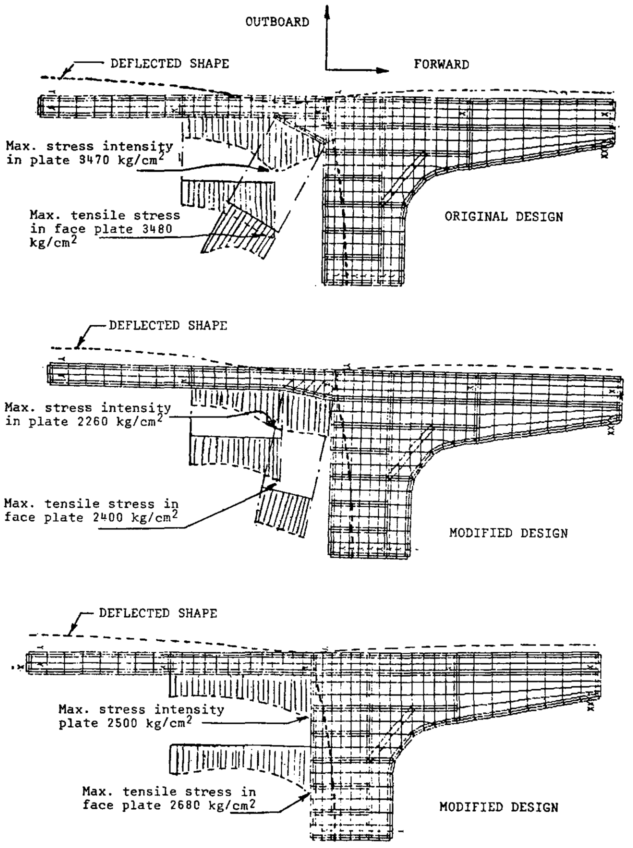

OUTBOARD
FORWARD
DEFLECTED SHAPE
Max. stress intensity in plate 3470 kg/cm^2
Max. tensile stress in face plate 3480 kg/cm^2
ORIGINAL DESIGN
DEFLECTED SHAPE
Max. stress intensity in plate 2260 kg/cm^2
Max. tensile stress in face plate 2400 kg/cm^2
MODIFIED DESIGN
DEFLECTED SHAPE
Max. stress intensity plate 2500 kg/cm^2
Max. tensile stress in face plate 2680 kg/cm^2
MODIFIED DESIGN

Figure 4—Effect of connection detail on stress concentration

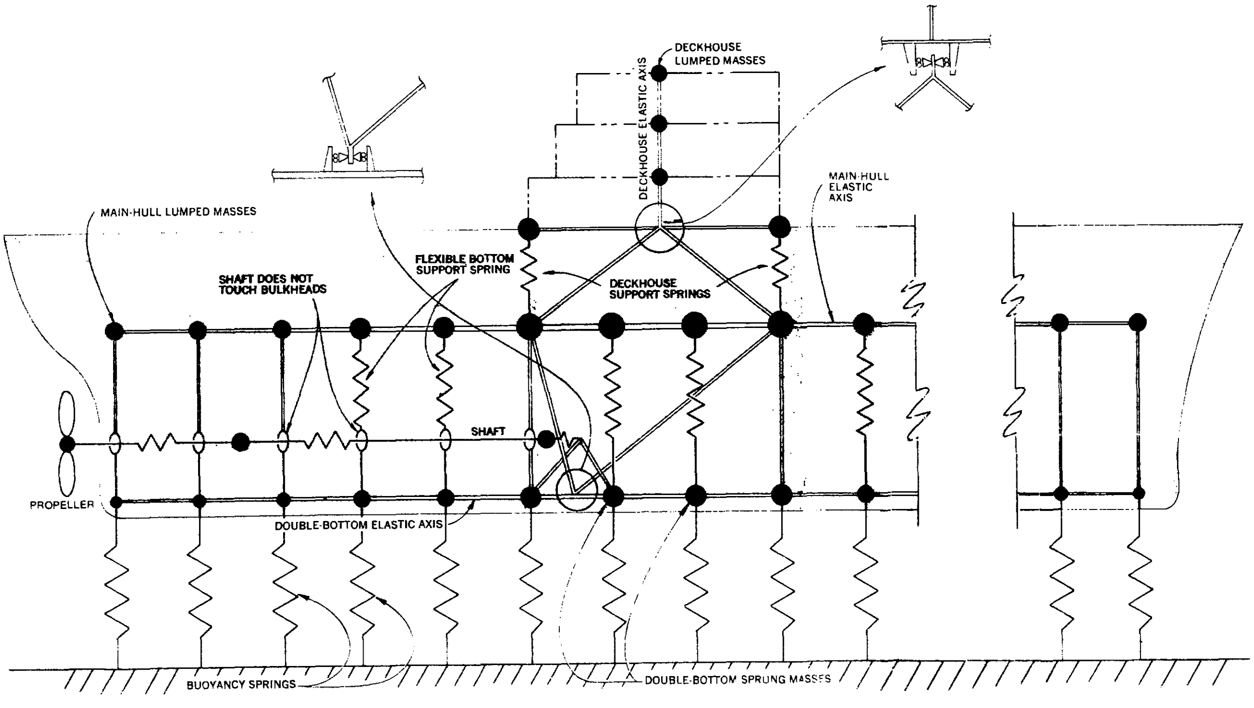

DECKHOUSE LUMPED MASSES
DECKHOUSE ELASTIC AXIS
MAIN-HULL ELASTIC AXIS
MAIN-HULL LUMPED MASSES
FLEXIBLE BOTTOM SUPPORT SPRING
SHAFT DOES NOT TOUCH BULKHEADS
DECKHOUSE SUPPORT SPRINGS
SHAFT
PROPELLER
DOUBLE-BOTTOM ELASTIC AXIS
BUOYANCY SPRINGS
DOUBLE-BOTTOM SPRUNG MASSES

Figure 5—Complete ship idealization

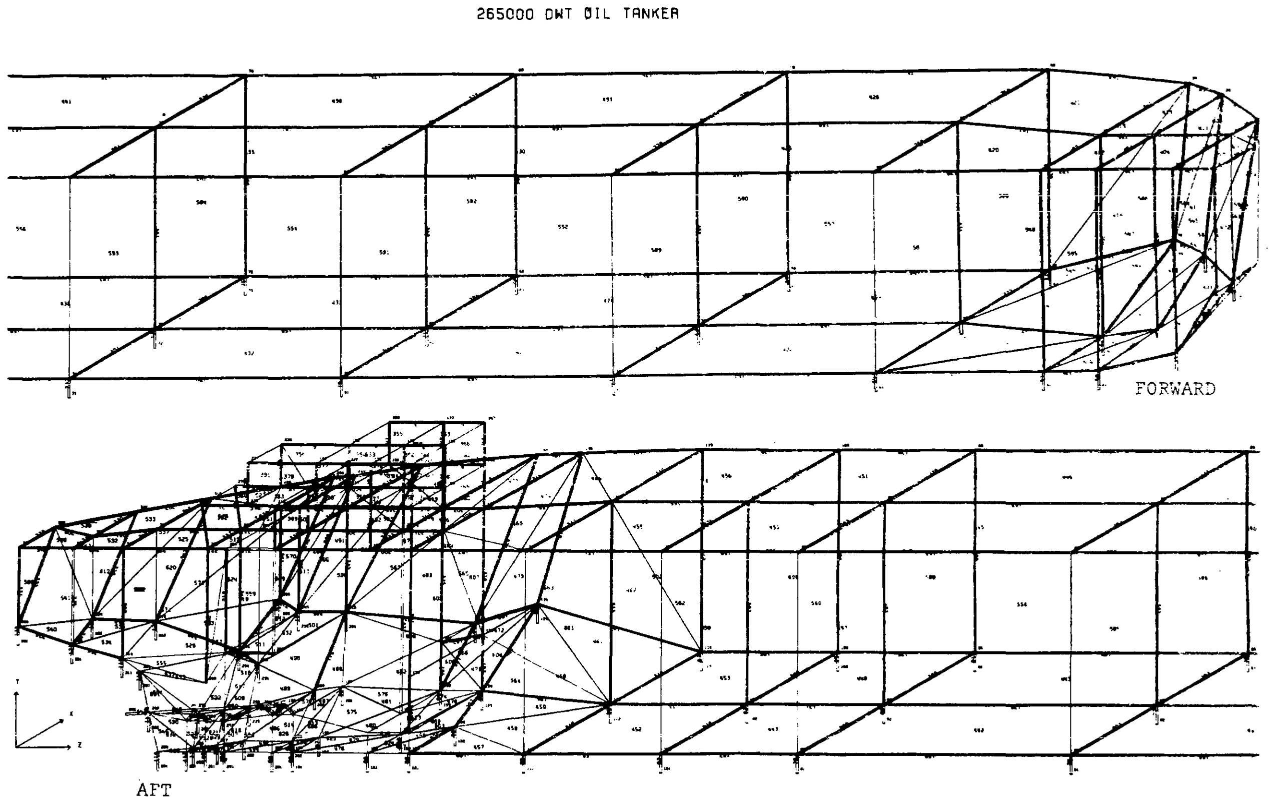


Figure 6—Isometric view of ship model (port)

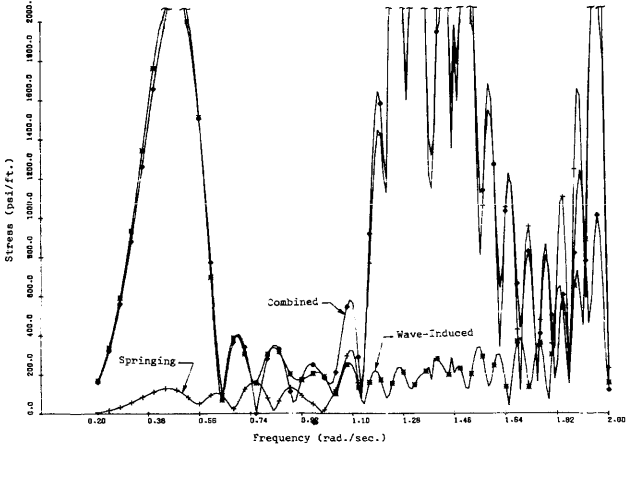


Figure 7—Dynamic midship bending stresses

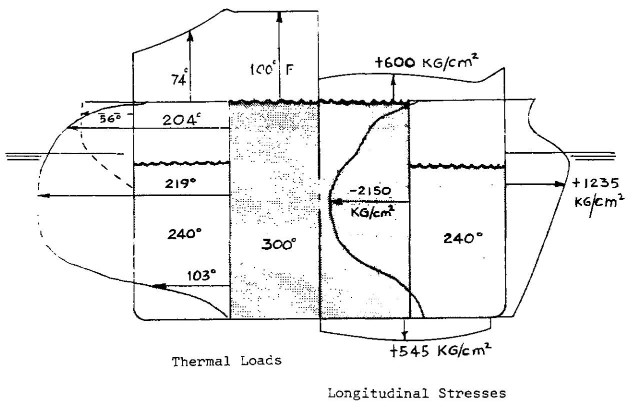
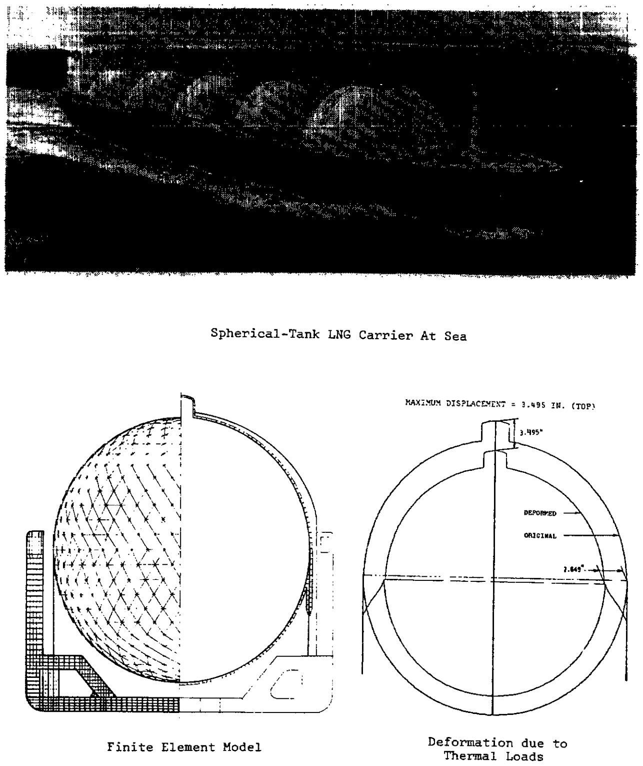


Figure 8—Thermal analysis—Asphalt carrier



Spherical-Tank LNG Carrier At Sea

Finite Element Model

Deformation due to Thermal Loads

Figure 9—Structural analysis including thermal effects—LNG carrier

300° F.[14] Figure 9 shows a picture of a spherical tank LNG carrier, a finite element model used for its analysis, and the thermal deformation of a portion of the tank consisting of the dome, the spherical tank shell and the cylindrical supporting skirt.[15]

NAVAL ARCHITECTURE PROGRAMS

Numerous computer programs have been developed over the years to perform various naval architecture calculations such as hull girder shear and bending moment, section modulus, hydrostatic stability and hull characteristics. These calculations all require a description of the hull geometry, usually in the form of offset data. Perhaps the most widely used and comprehensive hull characteristics program is SHCP (Ship Hull Characteristics Program).[16] This program develops both intact and damage stability characteristics for ship forms by conventional methods. Hull girder shear and bending moments can also be calculated. Computer-generated plots can also be produced, such as Figure 10, showing hydrostatic curves, and Figure 11, showing the transverse sections of the hull outline of a vessel.

In addition to comprehensive computer codes such as SHCP, special purpose programs for calculating grain stability, tank ullage, and section modulus are also available to naval architects. As a description of the hull form is a requirement for all of the above calculations, establishment of a common data base for the hull surface geometry is highly desirable.

Computer programs based on the classical naval architecture approach to stability of ship hulls are not easily adaptable to offshore drilling rigs such as the semi-submersible and self-elevating types. These rigs are generally composed of assemblages of tubular sections and/or geometric shapes having planar surfaces. Use of conventional offset data to describe their unusual shapes can be difficult and complicated. To handle drilling rigs, stability computer programs have been developed to perform intact and damage stability calculations for mobile drilling rigs. To overcome difficulties in describing the unusual and multi-connected shapes of these rigs, a typical program such as the ABS-developed DRILRIG[17] contains a library of solid element shapes (cylinders, tetrahedrons, hexahedrons, hemispheres, etc.) which are readily defined with a minimum of data. The user selects the shapes and assembles them so as to represent the compartmentation and geometric configuration of the rig. Stability calculations are then performed by conventional methods.

CONVERSATIONAL PROGRAMS

Conversational programs, which provide for back-and-forth communication between the engineer and the computer, have been successfully applied to obtain ship scantlings (structural properties, such as plate thicknesses and beam sizes) based on classification society rules.

Such a typical computer approach to ship design is the ABS/RULESCANT (**RULE SCANT**lings) systems of time-sharing programs that determine scantlings satisfying the ABS Rule requirements for midship sections of oil, ore or bulk carriers.[18] The input typed by the user at a time-sharing terminal consists of the basic configuration of the vessel, the plate and stiffener sizes and the location and arrangement of structural members. A data base (file) of all the processed ship information is created for use by the many output subprograms which can be individually selected for execution by the user at the terminal.

One of the programs is used to check scantlings of the midship section. The program determines the Rule-required plate thicknesses and stiffener and hull girder section moduli. It also calculates the weight per unit length, and the allowable shear stress and shear force. The formulas used by the program in determining the Rule requirements are also listed, as a matter of information to the user. After viewing the results, the user can dynamically change any of the input, including the given values of the plating and

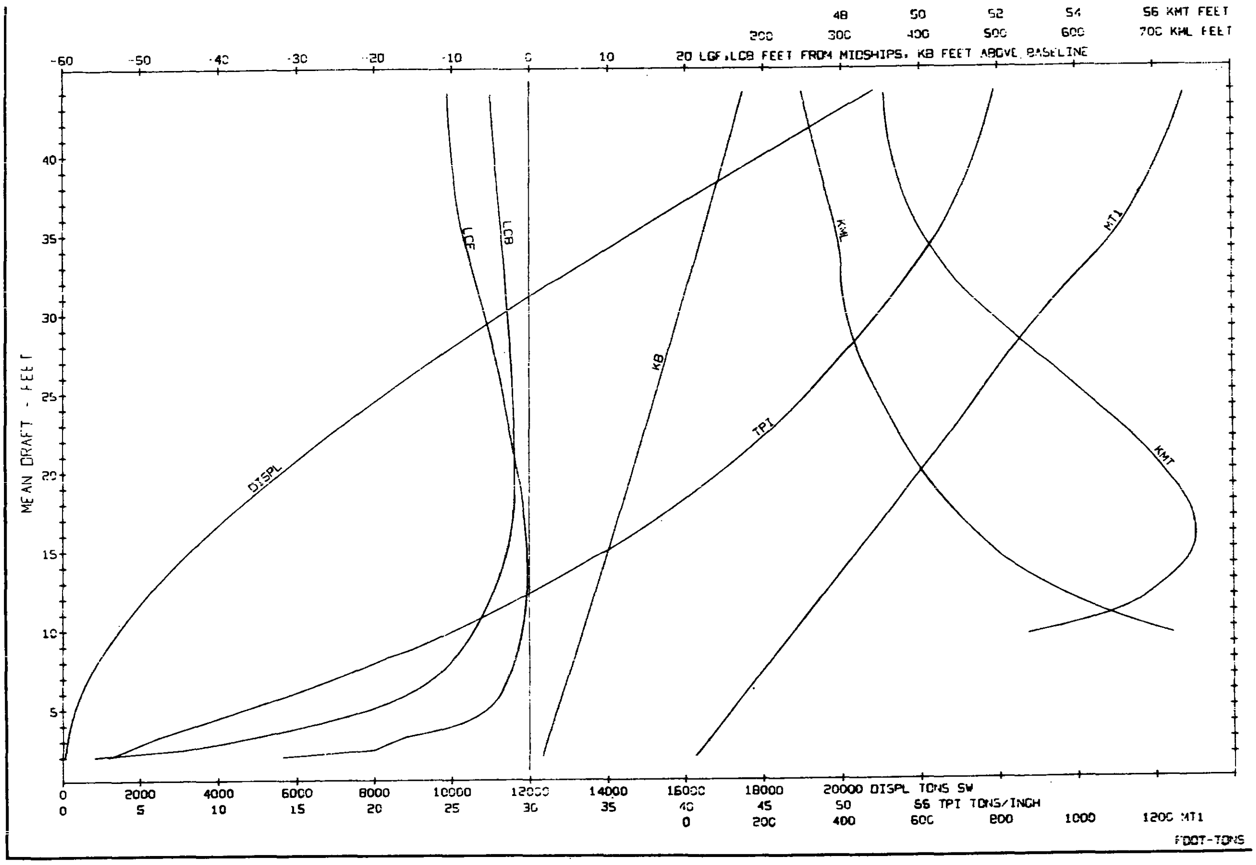


Figure 10—Computer plot of hydrostatic curves

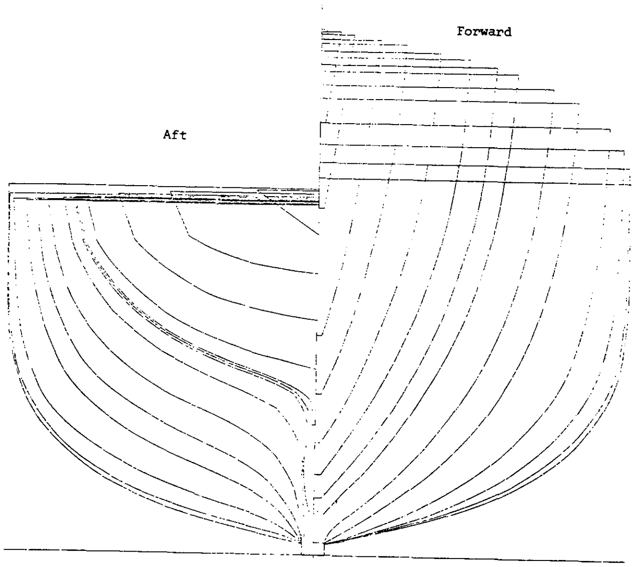


Figure 11—Body plan—Transverse sections of hull outline

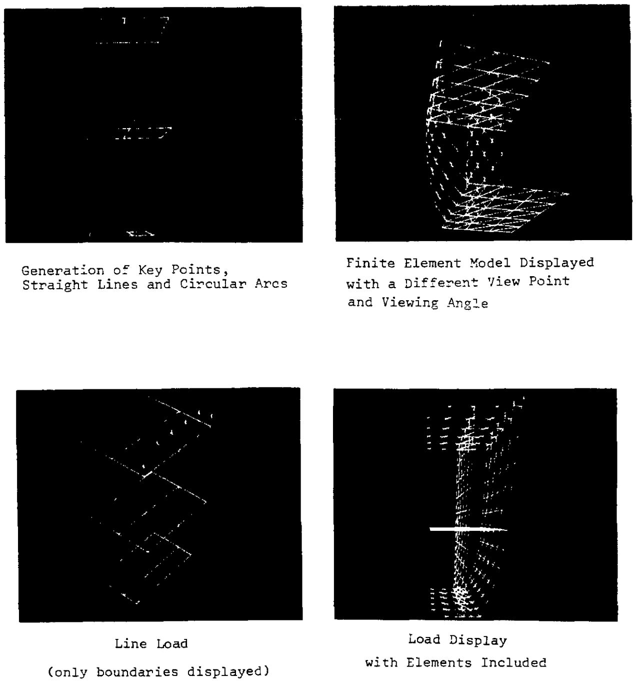
stiffener sizes. Then he can rerun the program while at the terminal and obtain answers within minutes of altering the input.

There is also a preliminary design subprogram, which requires input of only a basic definition of the midship geometry. The program then determines the minimum Rule requirements for the midship section, which can be used for preliminary design purposes.

Many more sections of the Rules will be integrated into a computerized system similar to ABS/RULESCANT.

COMPUTER GRAPHICS

With the improvement of the general purpose structural programs to analyze large and complex structures economically, the need for efficient methods of checking input data and reviewing output results becomes more pressing.[19] The field of computer graphics satisfies this need by producing visualizations of the structural



Generation of Key Points, Straight Lines and Circular Arcs

Finite Element Model Displayed with a Different View Point and Viewing Angle

Line Load (only boundaries displayed)

Load Display with Elements Included

Figure 12—Graphical displays of ship segment

models and stress patterns. The inherent advantages and disadvantages of the two basic methods of computer graphics, passive and interactive, dictate the usefulness and areas of application of each method.

Passive graphic systems include plotters (flatbed, drum, and electrostatic) and microfilm recorders. The nature of these devices precludes user interaction, and therefore they are best suited to applications where a user has time to review the resultant plots before making any changes or going on to the next step. The most popular applications include plots of the geometric model (input) and deflected shapes and stress contours (output). Since turnaround requirements for these applications can usually be measured in hours, the normal operating procedure is to run these jobs in the batch mode on large computer systems and produce tapes which can subsequently run on the plotters.

Most of the large finite element programs have plotting capabilities as part of the basic program or as separate add-on modules. In addition, there are many general purpose plotting programs that can use the output files generated by most finite element programs, although in some cases interface programs must be written. Input geometrical plots (two- and three-dimensional perspective views) and a wide variety of output plots (deflected shapes, force and moment diagrams, stress contours) form the most generally available plot features.

The basic advantage of the passive system is that large amounts of data can be processed economically. The plotter is the only additional equipment needed, and the actual computer runs necessary to generate the plots can be run in the batch mode, with minimal impact on a large computer system. The obvious disadvantage is the lack of interaction. Incremental mode plotters are relatively slow (detailed plots can take many hours), but electrostatic plotters can produce hard copies at rates comparable to most reproducing machines.

Interactive graphics systems consist of display consoles, means of entering and editing data (usually cathode ray tubes, tablets, keyboard devices), and a computer system to maintain the data files and perform the calculations needed to produce the plots.

Interactive systems find their greatest use in design work. The designer is able to communicate with the computer, see the results, and make the necessary changes. The earlier systems required either a totally dedicated medium-sized-computer or a large portion (partition) of the resources of a large computer. In recent years, the availability of time-sharing systems and powerful minicomputers has relaxed these requirements.

The GIFTS (Graphics-oriented Interactive Finite Element Time-Sharing Package) system,[20] designed primarily for ship structures, is one widely used interactive package. The entire system accesses a Unified Data Base (UDB) which stores all pertinent data on a set of random access files. Each individual module can access and operate on the UDB. After the entire model has been verified, part of the UDB forms the input to a general purpose analysis program such as DAISY, NASTRAN (NAsa STRuctural ANalysis),[21] or SAP IV. The output from the analysis program is then incorporated into the UDB and additional modules can display results.

Some of the displays obtained during the various phases of the analysis of a vessel are shown in Figure 12.

INFORMATION RETRIEVAL SYSTEMS

The service experience of vessels at sea provides a wealth of data to evaluate ships. Information on occurrences of structural and mechanical damage and failures must be collected and statistically analyzed. Subsequent feedback of this information will enable shipbuilders and designers to incorporate this knowledge into future improvements in design, construction and analysis techniques. The only means to efficiently handle the vast amounts of data involved is through information handling systems.

A comprehensive computerized system for information storage, correlation and retrieval has been implemented at ABS. This system, known as ABSIRS (American Bureau of Shipping Information Retrieval System)[22] handles data concerning shipowners, shipbuilders, ship characteristics, service histories and other pertinent data relevant to merchant vessels of the world. ABS uses its vast stores of information and worldwide telecommunication facilities to provide easy and rapid accessibility to the industry.

The ABSIRS system consists of seven files, whose functions can be summarized as follows:

Master File is the nucleus of ABSIRS. It contains all the data in the published RECORD of the American Bureau of Shipping,[23] a ship registry that contains pertinent characteristics of more than 42,000 vessels—virtually all sizeable vessels in existence—including the more than 13,000 vessels that have been classed by ABS.

Technical Notes File contains service data limited to statistical compilations and correlations of damages or casualties in vessels classed with ABS since 1965. Hull or machinery failures that are considered significant are entered in the file, i.e., those failures that may eventually show a recurring problem and thereby prompt Rules changes or revisions in existing construction techniques. The File contains a brief description of hull damages according to type (buckling, welding, cracks, corrosion, etc.).

Construction File stores additional information on vessels of ABS classification, including particulars on characteristics of hull construction and materials and machinery items and associated components.

Dead File stores data on ships that have ended service life. The recorded data passes from the Master File to the Dead File, insuring a preservation of the vessel's history.

On-Order File carries data on vessels in excess of 1,000 gross tons that are either being constructed or are under contract to be built in shipyards around the world.

Owners File is a record of the names and addresses of owners, agents, and operators of ABS-classed vessels appearing in the RECORD of the American Bureau of Shipping.

Shipbuilding and Drydock File lists the names, locations, capacities, and descriptions of shipbuilding, drydock, and repair facilities available throughout the world.

Each file can be searched separately and there is a multi-file capability that allows the information within each file to be cross-correlated with data in any other file or files. Inquiries may be based upon any category stored in the computer, and many report formats are available to suit the user's needs.

CONCLUSIONS

Computer usage in the marine industry is extensive, diversified and expanding. It combines the use of mathematical techniques with the latest technical facilities of computers, software, hardware and communication with the user.

The developments of computer analysis in the industry point to the following future trends:

1. More intelligent use of the computer as an engineering tool will result in more economical analysis methods. For example, advanced techniques such as substructuring will find increased applications in the design and analysis of ship structures.
2. Increased use of computer graphics display terminals will facilitate the automatic generation and verification of the large amounts of data necessary to create an extensive finite element model.
3. Engineering analysis programs will acquire a more interdisciplinary character. For example, the hydrodynamic forces acting on a ship will be calculated by the same computer program that determines the resulting structural response.
4. More interactive programs emphasizing user-computer interaction will result in a greater variety of applications among a wider spectrum of users.

REFERENCES

1. American Bureau of Shipping, *Rules for Building and Classing Steel Vessels*, 1975.
2. Kamel, H. A., D. Liu and S. G. Stiansen, "Naval Transportation," presented at the *Society of Enginering Science, Inc., 12th Annual Meeting*, Austin, Texas, October 1975.
3. Moe, J., "The Finite Element Technique—A New Tool in Structural Analysis," *Finite Element Methods in Stress Analysis*, edited by I. Holand and K. Bell, Tapir, Trondheim, Norway, 1969.
4. Kamel, H. A., D. Liu and E. I. White, "The Computer in Ship Structure Design," *Numerical and Computer Methods in Structural Mechanics*, edited by S. J. Fenves, N. Perrone, A. R. Robinson and W. C. Schonbrich, Academic Press, New York, 1973.
5. Clough, R. W. and K. J. Bathe, "Finite Element Analysis of Dynamic Response," *Advances in Computational Methods in Structural Mechanics and Design*, edited by J. T. Oden, R. W. Clough and Y. Yamamoto, UAH Press, University of Alabama at Huntsville, 1972.
6. American Bureau of Shipping, Research and Development Division, *DAISY Users Manual*, 1975.
7. Elbatouti, A. M., D. Liu and H. Y. Jan, *Structural Analysis of SL-7 Containership under Combined Loading of Vertical, Lateral and Torsional Moments Using Finite Element Techniques*, Ship Structure Committee Report SSC 243, 1974.
8. American Bureau of Shipping, Research and Development Division, *Structural Analysis of 130,000 M³ LNG Carrier*, Technical Report RD-73006, December 1973.
9. American Bureau of Shipping, Research and Development Division, *Structural Analysis of Longitudinal Stiffener Connections to Transverse Bulkheads*, Technical Report RD-75010, April 1975.
10. Kline, R. G. and E. U. Shipe, *Users Guide for Simulated Ship-Hull Vibration*, U.S. Steel Computer Program Documentation, 1971.
11. American Bureau of Shipping, Research and Development Division, *Vibration Study of 265,000 DWT Oil Carrier—Part 1—Analysis and Correlation*, Technical Report RD-75001-1, July 1975.

12. Bathe, K. J., E. L. Wilson and F. E. Peterson, *SAP IV—A Structural Analysis Program for Static and Dynamic Response of Linear Systems*, Report No. EERC 73-11, College of Engineering, University of California at Berkeley, April 1974.
13. Ueda, Y. and T. Yamakawa, "Thermal Nonlinear Behavior of Structures," *Advances in Computational Methods in Structural Mechanics and Design*, edited by J. T. Oden, R. W. Clough and Y. Yamamoto, UAH Press, University of Alabama at Huntsville, 1972.
14. American Bureau of Shipping, Research and Development Division, *Thermal Stress Analysis of Oil Tanker for Carrying Hot Asphalt*, Technical Report RD-73004, August 1973.
15. American Bureau of Shipping, Research and Development Division, *Structural Analysis of 125,000 M^3 Spherical Tank LNG Carrier*, Technical Report RD-73001, April 1973.
16. Naval Ship Engineering Center, Department of the Navy, *Ship Hull Characteristics Program—SHCP Users Manual*, CASDAC #231072, January 1976.
17. American Bureau of Shipping, Research and Development Division, *DRILRIG Users Manual*, October 1975.
18. American Bureau of Shipping, Research and Development Division, *RULESCANT Users Manual*, July 1975.
19. Stiansen, S. G., "Structural Response and Computer-Aided Design Procedure," presented at the *SSC-SNAME Ship Structure Symposium*, Washington, D.C., October 1975.
20. Kamel, H. A. and M. W. McCabe, *A Graphics Oriented Interactive Finite Element Time Sharing Package* (*GIFTS*), Office of Naval Research Report, Contract No. N00014-67-A-0209-0016, July 1973.
21. Butler, T. G. and D. Michel, *NASTRAN—A Summary of the Functions and Capabilities of the NASA Structural Analysis Computer System*, National Aeronautics and Space Administration, NASA SP-260, 1971.
22. Mole, K. M., "Computer Usage at the American Bureau of Shipping," presented at the *International Conference on Computer Applications in the Automation of Shipyard Operation and Ship Design*, Tokyo, Japan, August 1973.
23. American Bureau of Shipping, *RECORD of the American Bureau of Shipping*, 2 Vols., 1976.

Evolution of automation in terminal air traffic control

by HOWARD R. McGLAUFLIN
Federal Aviation Administration
Burlington, Massachusetts

ABSTRACT

The Evolution of Automation In Terminal Air Traffic Control was written to provide the reader with an overview of the why, when and how of the needs of air traffic controllers in the terminal. This paper compares systems and a decision to award contracts. This paper contains a description of the subsystems used at ARTS III facilities and its capability for expansion. Attachments to this paper show a radar display, an ARTS III display, a system design and a work flow-through design, basically how it works.

The intention of this paper is to provide an overview of the evolutionary process of system automation in the Terminal Air Traffic Control environment, and provide a look at the basic flow of data through the existing system.

To do his job in the 1920's, the controller had only to wave a green or red flag; in the 1930's he operated a radio in a control tower; in the 1940's he had to learn to operate a radar set; and now he must start learning to communicate through a computer.

The need for some degree of automation to assist controllers in Terminal Air Traffic Control has been apparent for some time. Air Traffic volume has been increasing and is forecasted to increase so that many terminals will be taxed to capacity. Delays at the busiest airports are frequent, and all too often, lengthy.

The formal initiation of plans to develop an automated system for terminal traffic control occurred in March 1961, when the Project Task Force was established. The result was the Project Beacon Report, which was submitted to the FAA (Federal Aviation Administration) and then to President Kennedy in September of 1961. One significant recommendation of this report was, "utilization of general purpose digital computers to provide controllers with aircraft position information."[1]

Subsequently, the FAA's System Design Team was established to fulfill the requirements of the Project Beacon Report; a functional specification for an experimental model of an automated radar terminal system that could be used for appraisal of the concepts in a field environment was developed. In 1963, the FAA awarded a contract to Univac to provide the computers, software, and system integration efforts for the establishment of this model known as Advanced Radar Tracking System I, in the Atlanta, Georgia, terminal. After field testing, this system was implemented and has been in use since 1966. Basically, Advanced Radar Tracking System I, utilizes a general purpose digital computer together with video digitizers and displays to increase the radar and beacon video on a controller's console. It does so by the display of alphanumeric aircraft identity and altitude information which is automatically associated with the proper video returns.

The next activity involving terminal air traffic control was the implementation of an automation system in the New York Common Instrument Flight Rule Room. This room was implemented in 1968, located at Kennedy Airport in New York City, and permits the individual airspaces over Kennedy, LaGuardia, and Newark Airports to be combined into a single operation by utilization of computer assistance. This system is called Advanced Radar Tracking System IA, "this system differs from the Advanced Radar Tracking System I in that it uses two separate, non-collocated radar and beacon inputs."[2]

In 1969, the FAA arranged for implementation and demonstration of a terminal automation system suitable for medium traffic load in Knoxville, Tennessee. The purpose was to demonstrate that a small, stored program computer offers more flexibility and capability than special purpose calculating devices, which had been devised for the detection and display functions at smaller installations. After these hardware implementations were carried out, in December of 1969, a committee was organized by the Department of Transportation (commonly called the 1980 Committee) Air Traffic Control Advisory Committee whose duty it was to project air traffic requirements until the 1980 and 1990 time frame. This Committee report was released and is now being used as guidelines for automation development and enhancement. One major recommendation of the 1980 committee's report affecting terminal automation was, "The addition of more capability to the terminal automation system in order to provide such functions as command and control sequencing, conflict detection, and collision avoidance, and other functions which can increase safety or maximize terminal system aircraft acceptance rate."[3]

Based on experience gained from the operation of ARTS I and IA, as well as continuing analysis of the present and projected air traffic situation, the FAA developed a design for an improved air traffic control system. The Advanced Radar Tracking System III, ARTS III, was designed to simplify the acquisition and maintenance of radar identification, display beacon derived altitude data, simplify intrafacility and interfacility coordination procedures and reduce the communications workload. (See Attachment #1) The result is a more efficient utilization of terminal airspace and air traffic control personnel and an enhanced safety system. The ARTS III system is now being implemented at 62 major air terminals, and at the FAA Academy in Oklahoma. It is intended to be a first step for many in terminal automation of air traffic control and to provide an automation basis on which to build. In order to provide the capability for growth of system functions at the same pace with increased requirements and resultant developments, ARTS III was additionally designed to be built with a modular expandable concept. Because software is, of its own nature, modular expandable when using the executive program/sub-program hierarchy, and also dependent on the hardware modularity, the primary design considerations were related to the hardware. Being an add-on system, basically, the major sub-systems were added to the existing terminal area equipment: Data Acquisition Subsystem (DAS), Data Processing System (DPS), Data Entry and Display Subsystem (DEDS).

The following is a description of the three subsystems used in the ARTS III system:

(1) The ARTS III DAS, which will accept inputs from a variety of airport surveillance radars and beacon interrogators which consist of the azimuth, range and timing group, the beacon reply group, and the azimuth pulse generator. The azimuth, range and timing group generates all the basic timing pulses for use by the DAS. The beacon reply group detects and interprets the beacon video signals; the azimuth pulse generator, physically mounted to the radar pedestal, converts radar antenna position to digital position data.

(2) The DPS consists of the data processor (IOP) and its peripheral equipment, as well as the operational computer program. (See Attachment #3) The IOP receives digitalized beacon target report messages from the DAS. It also receives flight plan data from a computer in the enroute center via the ICA. The IOP also receives controller-initiated messages from the DEDS. The IOP provides the system with a capability for arithmetic computation, logical decision making, data processing, and overall system coordination. Operation of the IOP is controlled by a stored program located in the memory bank with each major processing task organized into a sub-program. The IOP then employs an executive control sub-program that without performing any processing tasks itself, serves to control the execution as needed, rather than in fixed sequence. The task selected depends upon an assigned priority scheme that adapts to changes in the system processing load and permits the processor to respond to simultaneous external demands. Some tasks are executed on the basis of a fixed time interval, while others depend upon the completion of prerequisite processing of external events. The IOP provides control of overall communications between the DAS, the DEDS, and the peripheral equipment via high speed digital input/output channels.

The basic DPS includes peripheral equipment commonly found in a data processing application:

(1) A magnetic tape unit provides permanent storage for the computer programs, and is used to load the programs into computer memory. In addition, selected data obtained during system operation may be recorded on magnetic tape for future processing and analysis.

(2) An input/output console containing a low speed printer, typewriter and keyboard, and paper tape facilities is provided for controller communications with the IOP. This console is used off-line to enter variable parameters required by the system and is used as a back-up device to load programs when the magnetic tape unit is unavailable. It also provides on-line printout capabilities for various sub-programs. The hard copy may provide alarm, recording on requested functions. Alarm printouts result from program-detected malfunctions in hardware and errors in input.

(3) The DEDS consists of a common equipment assembly, display consoles, and data entry sets. The DEDS provide the man-machine interface between the air traffic controllers and the automation equipment.

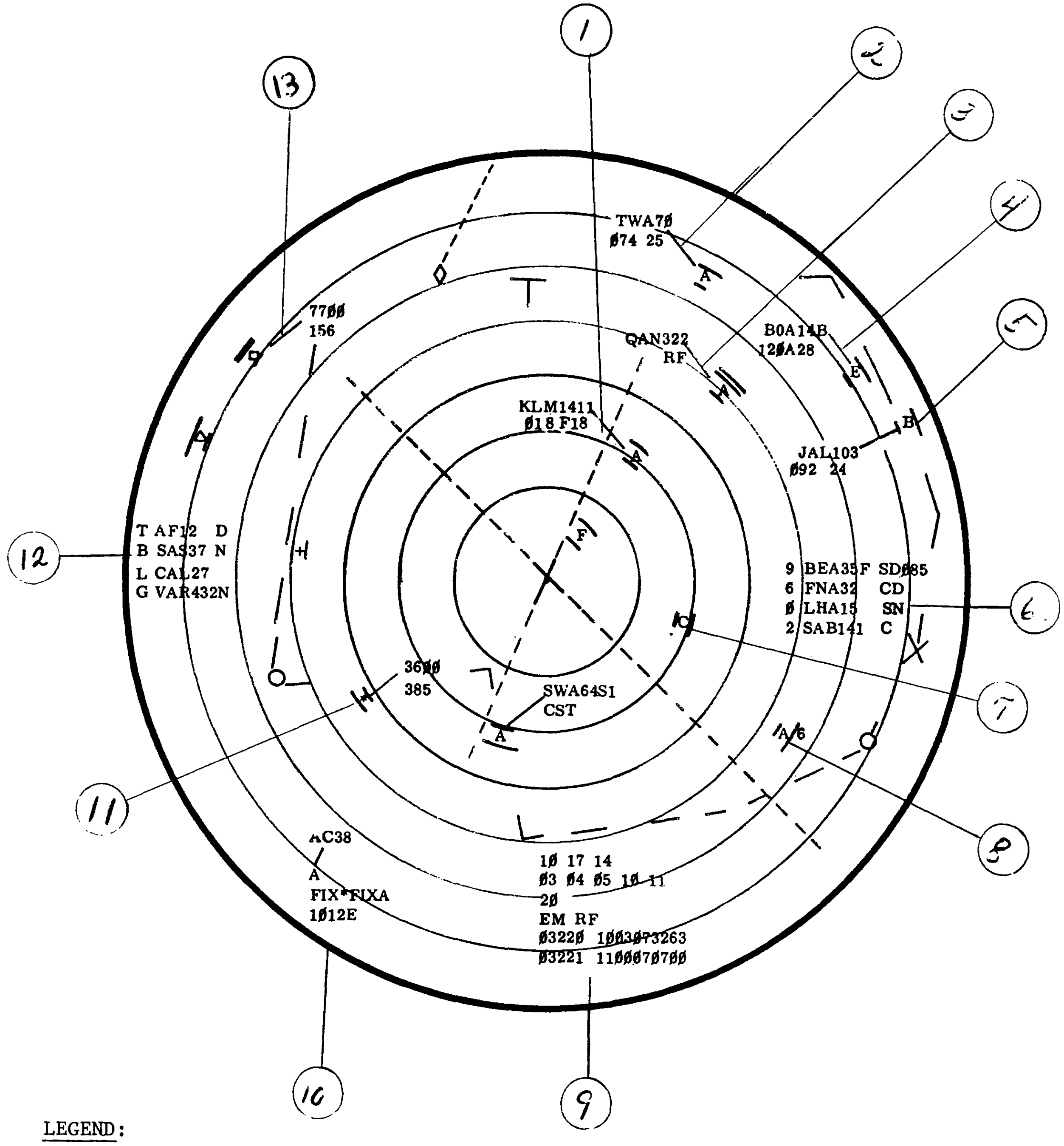
Since the ARTS III system has been in operation at Chicago O'Hare Field since May of 1971, many questions have arisen and one point has been made very clear, "the automation applications made to date in the air traffic control system do not seem to have increased the capacity and productivity of the system, even though there are some indications of increased safety and relief in the workload of the controller."[4]

This paper has attempted to provide an overview of the evolutionary process of system automation in the Terminal Air Traffic Control environment.

BIBLIOGRAPHY

1. "An Analysis of Project Beacon," United States Federal Aviation Agency, Air Traffic Service, Washington, D.C., November, 1962.

2. Anderson, R. H., "Data Processing in the New York Common IFR Room," *Sperry Rand Engineering Review,* 1967.
3. "Report of Department of Transportation Air Traffic Control Advisory Committee," Volume I, Dept. of Transportation, December 1969.
4. Hill, J. H., "Advanced ATMS and Controller Responsibility," *The Journal of Air Traffic Control,* January/February, 1973.
5. "*ARTS*" Sperry Rand, PX6508, July 1971.
6. *Sperry Rand Engineering Review,* Volume 24, No. 2, 1971.
7. White, Peter T., "Behold the Computer Revolution," *National Geographic,* Volume 185, No. 5, November 1970.



LEGEND:

— Primary Radar Return

—— Beacon Radar Return

═ Emergency Beacon

Alpha-Numeric Display

Tracked targets, indicated with alphanumerics, may be offset to any 45° direction. Untracked targets are always offset to NE.

Attachment #1—A Representation of the ARTS III Vdieo Screen (Alphanumeric Display)

1. KLM 1411: Tracked handoff from Controller A to Controller F (Format blinks after F accepts)

2. TWA 70: Tracked by Controller A full data block displayed. (Alt:7400 ft., ground speed 250 kts.)

3. QAN322: Tracked by Controller A, Radio failure (RF blinks)

4. BOA14B: Tracked by Controller E at enroute center, handoff from Controller E to Controller A (format blinks).

5. JAL 103: Tracked by Controller B, "Quick Look" by Controller A caused the data format to appear.

6. Coast/suspend list - provides identification of aircraft under control of Controller A but which are not currently being tracked.

7. "C" : tracked by Controller C no data format appears since target is being controlled at another display.

8. "A6" : tracked by Controller A target has been assigned temporarily to a "Suspend" status as indicated by line identifier "6".

9. System Data Area - Provides current time, altimeter setting, selected beacon codes untracked emergency/radio failure indicators and memory readouts.

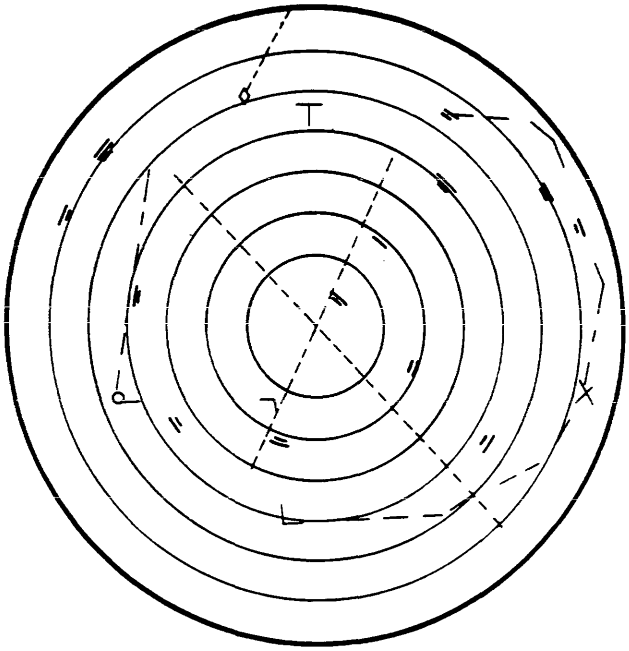
10. Preview Area - Provides keyboard entry characters and controller requested data (e.g. flight plan data).

11. * Untracked, limited data block.

12. Arrival/Departure List - Provides identification of aircraft which are scheduled to arrive at the entry fix or depart the airport within a few minutes and will be controlled by Controller A.

13. Untracked, emergency (format blinks).

Attachment #1A—Legend of Alphanumeric Display



Non-Alpha-Numeric Display

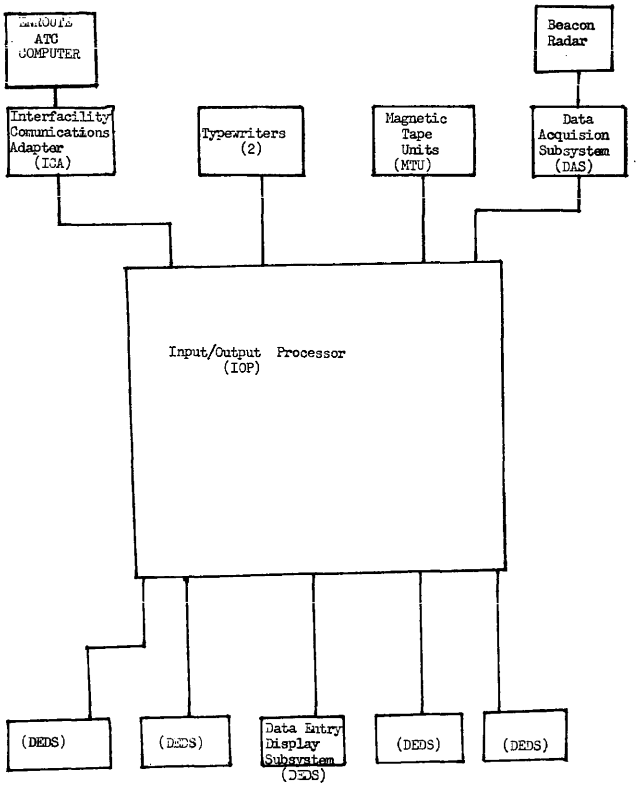
LEGEND

▬ Primary Radar Return

— Beacon Radar Return

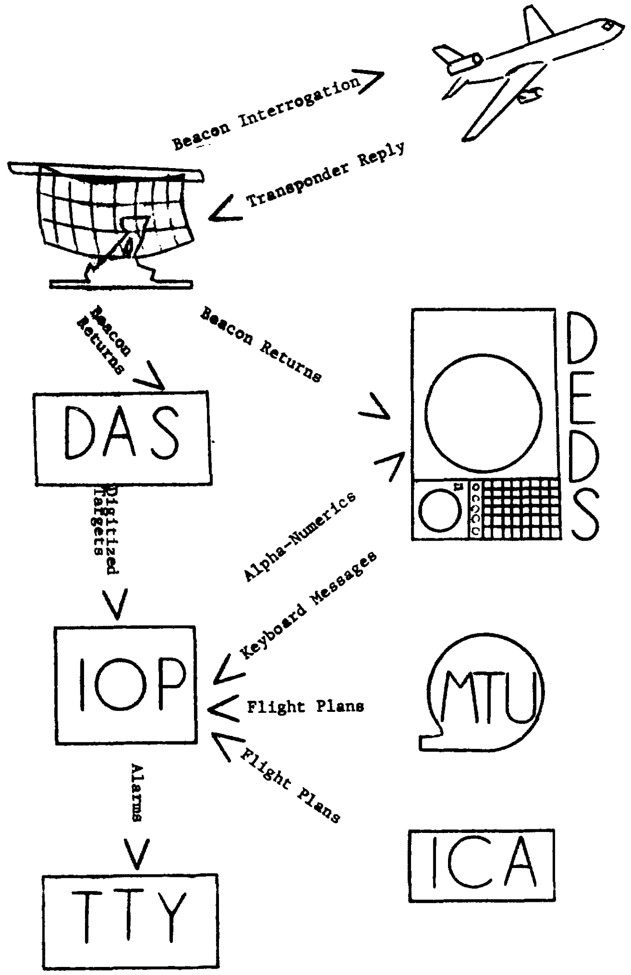
═ Emergency Beacon

Attachment #2—Non-alphanumeric Display


ENROUTE ATC COMPUTER
Beacon Radar
Interfacility Comunications Adapter (ICA)
Typewriters (2)
Magnetic Tape Units (MTU)
Data Acquision Subsystem (DAS)
Input/Output Processor (IOP)
(DEDS)
(DEDS)
Data Entry Display Subsystem (DEDS)
(DEDS)
(DEDS)

Attachment #3—Block Diagram of DPS and related equipment prepared by H. R. McGlauflin, Plans and Programs Specialist, Air Traffic Division, Burlington, MA

The following illustration shows the basic flow of data through the system.



Attachment #4—Basic Flow of Data Through the System, prepared by T. Pastore, Secretary, Boston Tower.

Computer graphics in an automatic aircraft landing system*

by E. H. REITAN
ITT Gilfillan
Van Nuys, California

and

S. H. SAIB
University of California
Santa Barbara, California

ABSTRACT

The Marine Air Traffic Control and Landing System (MATCALS), being implemented by the Naval Electronic Systems Command, provides advanced capabilities for fully automatic all-weather landing. Using radar-derived aircraft position reports, a ground computer provides appropriate guidance commands, which are transmitted to the aircraft's autopilot, and used to fly the aircraft automatically to touchdown.

Due to severe system requirements of one-half minute landing intervals, and six aircraft simultaneously on final approach, a unique combination of display presentation and operator interaction techniques must be used. These support ground operators who as controllers are responsible for initialization of each landing sequence, monitoring its progress, and aborting the sequence if an unsafe condition develops.

A display concept has been developed for MATCALS with the goal of reducing the controller's workload and increasing his effectiveness. The display is used as a single working surface for both output and input functions. A dynamic, graphical display format presentation, with alphanumeric annotation, and alert information is displayed for the operator in multiple colors. More than 150 system controls are organized on the same display as a highly structured hierarchy of virtual control buttons grouped into menus. The operator is prompted through all data entry and control sequences. All operator entry is made by using a "Rand" type data tablet.

MATCALS OVERVIEW

A simplified block diagram of MATCALS is shown in Figure 1. MATCALS is organized into three main system segments: the Air Traffic Control Segment, the All-Weather Landing Segment, and the Control and Central Segment.

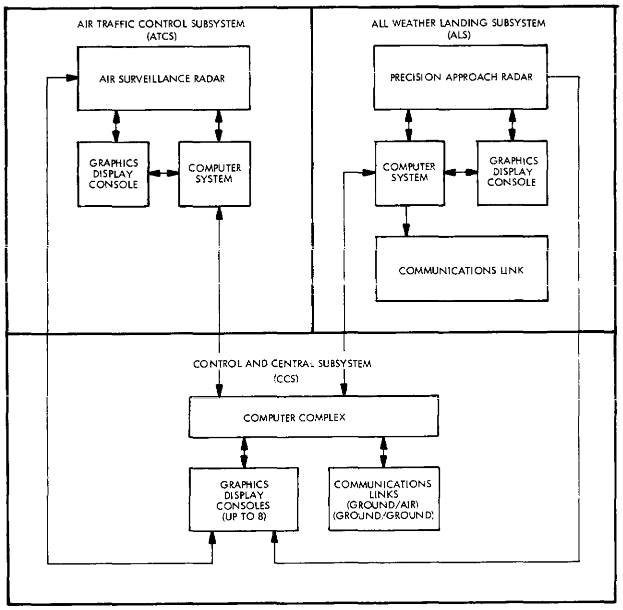


Figure 1—MATCALS system overview

The system normally operates from the Control and Central Segment which receives information from the two other segments. The Air Traffic Control Segment provides automated surveillance and traffic control within 60 miles of the airfield. Specifically, MATCALS provides for surveillance, identification, tracking, sequencing, vectoring, and inter-facility coordination for all approach, departure and overflight operations within the terminal area.

The MATCALS Landing Segment provides for fully manual to fully automatic landings. The sensor for

* The comments made herein are those solely of the authors since they pertain to early research and development and do not necessarily represent exact Government or technical requirements of the United States Marine Corps and the Naval Electronic Systems Command.

the landing segment is the AN/TPN-22 radar system shown in Figure 2. This precision approach radar provides the necessary positional reports with sufficient accuracy for automatic landings.

Three modes of landing are available to an aircraft. In Mode 1, ground derived steering commands are transmitted over a data link and directly introduced into the aircraft autopilot.

Under Model I, the system is able to transmit course deviations to any cross pointer equipped aircraft via a variety of data links. Specific up-links which have been implemented include pseudo-ILS signals receivable on standard airborne ILS equipment and side-band coded signals on a UHF voice channel. Pseudo-MLS signals receivable on the planned standard MLS airborne subsystems will be implemented later.[1]

In Mode III, ground derived approach course deviations are displayed to the landing controller who verbally transmits the landing correction commands to the pilot. The Landing Segment also provides for maintaining inter-aircraft separation minimums, monitors the approaching aircraft for acceptable positioning within defined boundaries, and provides for automatically aborting unsafe landing sequences.

CONTROLLER DISPLAY DESIGN OBJECTIVES

Emphasis has been placed on the selection of the computer graphics display formats and the type of operator interaction mechanisms which will support the MATCALS Landing System display requirements.

The display system must provide for the monitoring and control of up to six simultaneously landing aircraft. Inherent in the concept is the facility for controllers to assume responsibility for additional aircraft control in the event of failure of other display consoles. The overall objective is reduction of controller's workload and increase of landing safety in a multiple landing aircraft environment. A number of assumptions were made and desirable features were identified concerning characteristics of operator abilities, the varying difficulty of the operator's task during downgraded conditions due to hardware failures and reduced manning support, and the operational mission of the MATCALS production system.

It is first assumed that the console users in general will have little or no computer training. It is therefore unrealistic to impose any unnatural input techniques or display format output techniques. The operator/computer interface has to be as natural as possible and tailored to his conception of the landing operation—not just to simplify computer requirements. Ideally, representation of data or conditions should be done pictorially using a situation display type of abstraction. Auxiliary information may be presented alphanumerically to reinforce the pictorial representation. This auxiliary data assists the operator with his evaluation of the present operational situation. It is felt that graphical presentations lend themselves to rapid qualitative operator evaluation of global relationships of the data, while alphanumeric data presentations amplify and quantify the details of the landing situation. Both types of presentation (viewed as hierarchy of information) are necessary, and are most desirable if presented simultaneously on the same display surface.

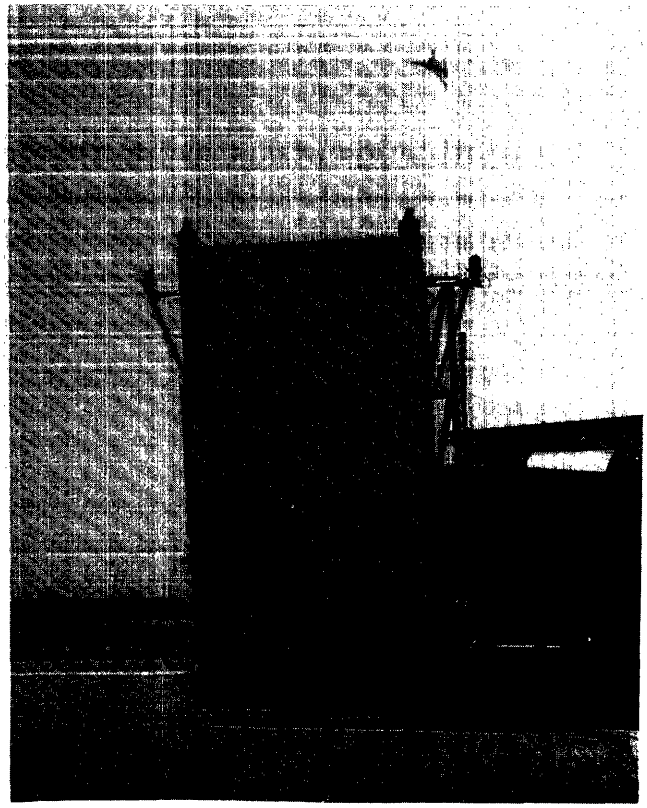


Figure 2—AN/TPN-22 precision approach radar system

The parallel presentation of pictorial (graphical) and tabular (alphanumeric) information simplifies not only the learning of the system but reduces the reaction time during critical and alert periods of system operation. The operator should not be required to divert his eyes to other presentation panels in order to make decisions, instead he should have available on a single display surface all pertinent information within his field of view. This allows him to assess the situation and make a corrective decision.

The complexity of the display format must be kept at a minimum. The state of the landing mission environment is presented as directly as possible. Information should be non-redundantly presented to the operator to assist and not hinder his evaluation of the situation.

All input mechanisms should be as direct as possible. Operator input should not be alphanumeric in nature (which is harder to understand being another level of abstraction). The MATCALS operational environment cannot tolerate lengthy input sequences such as from a physical A/N or function keyboard. Inputs instead should be implemented by operator selection of commands and controls from a list of alternatives presented on the display. This list must contain the complete set of controls for system operation. The range of his possible input responses to the output data may be limited by the computer system to those which are correct for that instance of time. The computer therefore guides the operator through entry sequences and prompts him as necessary for correct sequences of data entry. Menu groups of virtual control buttons are organized as a hierarchy into a tree. The operator progresses through the branches of the tree for all control and data entry functions. At any point in time, the interaction sequence is clear to the operator. In addition, the interaction method of implementation should be flexible to system enhancements.

Both output and input functions are performed through a single display working surface. All input controls are immediately adjacent to the output information from which a control decision must be made. Also, input controls are immediately available for operator actions. This varies from the conventional approach in that it eliminates additional hardware control panels and switches which add to the system complexity. The operator does not have to divert his attention from the single working surface display to search and locate other hardware controls.

Finally, the display must be responsive to operator requests. This requirement is satisfied by a careful design of both the hardware and software system. The level of responsiveness may vary, however, with the operator's evaluation of the complexity and importance of the requested service. As an example, the execution of a command to abort a landing sequence (a wave-off) should be immediately serviced and feedback acknowledging the action returned to the operator in a fraction of a second. Conversely, the request for a change of the display scale range may be delayed for several seconds with little operator dissatisfaction.

SYSTEM CONFIGURATION

Figure 3 is a simplified block diagram of the MATCALS System at Patuxent River, Maryland showing major hardware devices. The system is configured around a Univac AN/UYK-7 general purpose computer with 48K words of memory (32 bit words, 1.5 μs cycle time). The AN/UYK-7 is a medium scale, single accumulator machine which has floating point hardware and four independent data channels. Software for the AN/UYK-7 computer was developed in the CMS-2 language operating under an executive configured from the Common Program.

The central computer receives aircraft position from the precision approach radar, an ITT Gilfillan AN/TPN-22 radar with a nominal coverage of 10 miles in range, 46° in azimuth, and 8° in elevation. The radar system itself contains a minicomputer and provides stand-alone capability for Modes II and III landings in the event of a failure of the central system.

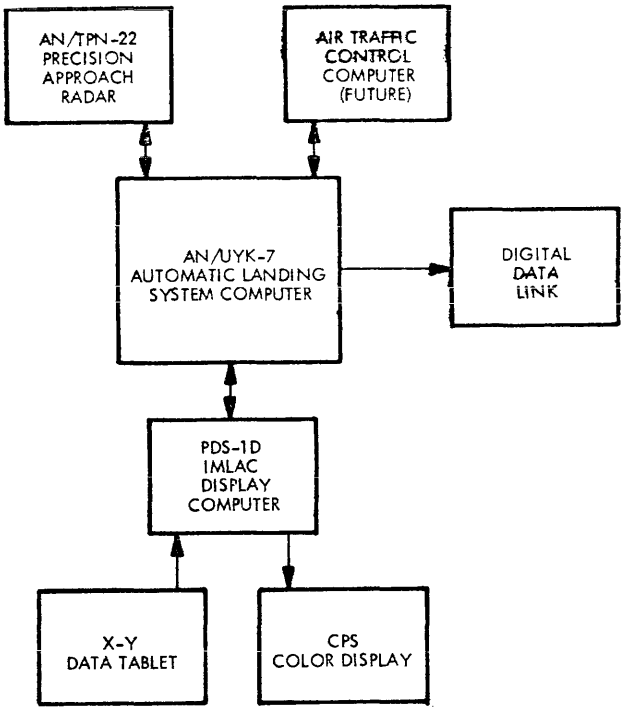


Figure 3—MATCALS landing system test bed configuration

The AN/UYK-7 computer calculates the flight path corrections and transmits the results to the aircraft via the digital data link. These corrections take the form of pitch and bank commands which are fed directly to the aircraft's control surfaces.

The IMLAC display computer with its associated CPS-8001 color slave display console and Computek data tablet constitutes the primary interface between the operator and the landing system. The IMLAC computer is a single accumulator machine with 16 bit words, 16K memory, 1.8 μs cycle time. The data tablet is utilized as the main interaction device. Unlike the conventional trackball, joystick, and lightpen, constant monitoring of the device by the computer cursor tracking calculation is unnecessary. There is no loss of track problem as with light pens, no wrap around problem as with trackballs, and no slow positioning as with trackballs and joysticks. Both functions of positioning and selection are provided by the stylus and tablet combination.

The division of responsibility between the central landing computer and the remote display computer was designed to

(1) reduce loading on a saturated AN/UYK-7 computer

(2) minimize the message transfer rate between the AN/UYK-7 and IMLAC computers
(3) provide for functionally partitioning display/interaction tasks from the landing guidance computer to allow easy extension for future multiple display support.

DISPLAY DESIGN

Design features

Based on the design objectives of the previous section, techniques were introduced which reduced the complexity of the display, assisted the operator in his interaction with the system, eliminated keyboard entries, and increased operator acceptance of the display format. These features were added without any degradation to the basic system requirements. The operational requirements for the MATCALS mission state that up to six aircraft may be simultaneously on final approach. The design therefore, has been predicated on a combination of display output and input techniques to allow the monitoring of the maximum number of aircraft by one operator, and a desired attainment of the control of six Mode I landing aircraft by a single operator.

Although it is doubtful that the single operator situation is desirable (with probably one operator for each two aircraft on a final approach being optimum), it is still mandatory that an operator have cognizance of all other aircraft on approach for safety reasons. The relationships of the aircraft under his control with those of other controllers must be continually monitored. In addition, it may also be postulated that due to hardware failures of other display consoles (and computer sections), or insufficient console manning resources, it is necessary that each operator can assume responsibility for more aircraft than is usual. This type of inherited fail-safe protection is a spin-off of the six aircraft status display requirement. The design of the MATCALS landing display format was therefore made with the assumption that data for all six aircraft on final approach be simultaneously presented, and that any individual operator can assume control of any or all aircraft.

Obviously this specification places an inordinate burden on the operator if conventional display and hardware techniques are utilized. Figure 4a depicts a conventional display which is used in the landing of a single aircraft. One could imagine the complexity of the hardware control panel if it were expanded to allow the landing of six aircraft. A system of this complexity would deluge the operator with controls, displays, and possible operational input combinations and confusing input sequences. On the other hand, Figure 4b shows the display hardware used in the MATCALS system to land six aircraft with just the single display surface and the tablet.

Display format design philosophy

The simpler hardware is possible due to the implementation of techniques wherein the computer organizes the type and format of data graphically or tabularly presented to the operator. The operator executes all data entry and request sequences to display auxiliary information, monitors the operation of the radar and computer hardware and software, analyzes and observes landing sequences, and is notified of any abnormal alert or danger conditions.

Four distinct types of information are presented on the display CRT face (Figure 5):

Graphical—Presentations with amplifying alphanumeric information
Tabular Parameter Data
Alerts and Status Information
"Virtual" Pushbutton Menus

Up to now, situation data of radar video presentations have been the conventional and common method of information presentation in aircraft control and aircraft landing display systems. To clarify the data on the screen, graphical and pictorial information has been added in some cases, often annotated by limited amounts of alphanumeric information (such as tracking data blocks for air traffic control systems).

The display format implemented for the MATCALS effort extends these concepts. In addition to graphical information, critical landing system parameters are presented grouped as tabular listings of alphanumeric data. Alert messages of hardware or software malfunctions and/or landing system safety irregularities are presented as necessary along with system status information as requested by the operator. Instead of hundreds of hardware control buttons continually being allocated space on a panel, the computer dynamically presents "virtual" control button panels which are displayed on the face of the CRT only when they are necessary and removes the panels when their use is no longer required. For example, it is necessary to have the controls which initialize various parameters to define an impending landing sequence (such as entering aircraft type and aircraft tail number) only at specific time intervals during the landing sequence. At other times these controls are extraneous. The display to be implemented as part of MATCALS utilizes this technique of control panel variability to simplify input operations and to reduce display format complexity.

The MATCALS Display presents all system status and controls in a single basic display format to the operator. Extending this concept to overlay the display graphical situation data with background air-search radar or precision approach radar video is practical using time compression display techniques. Whereas older conventional displays commonly used round CRT's, the MATCALS display organization, using multiple output data types, calls for a rectangular display

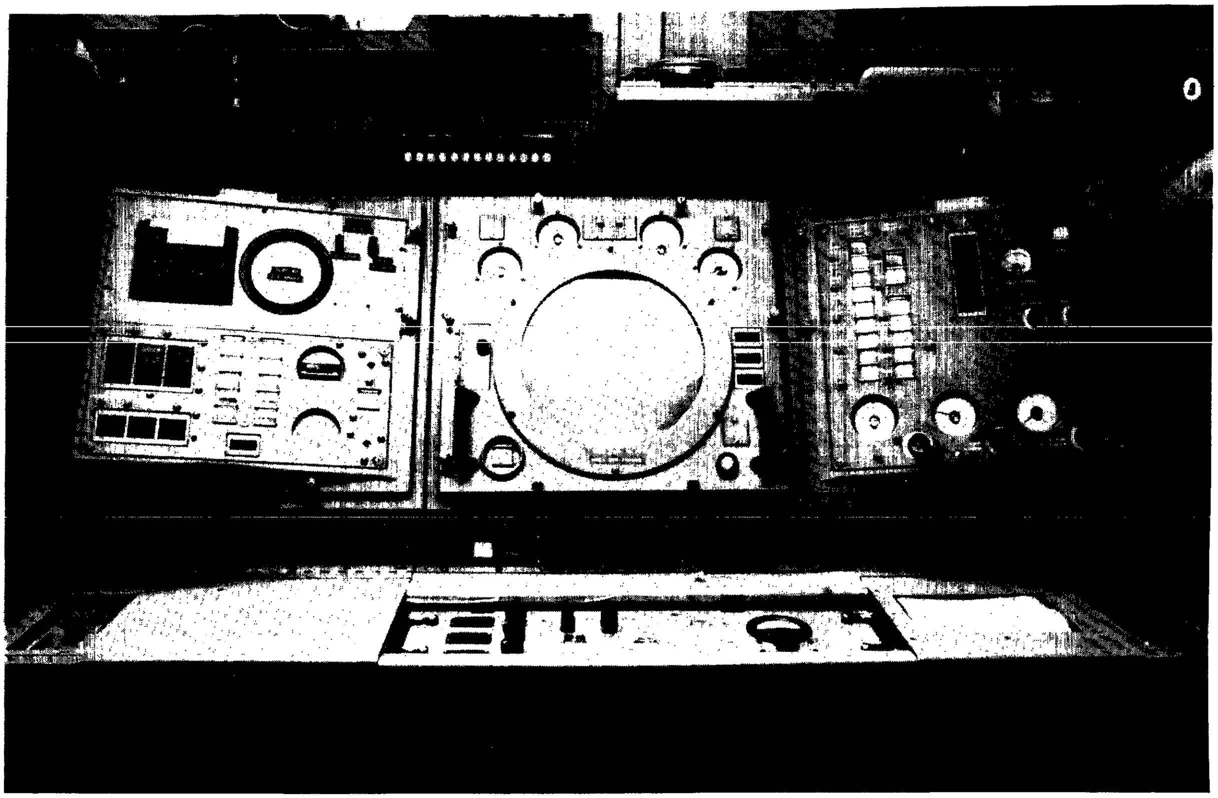


Figure 4—Evolution of landing controller operational consoles
a—Older conventional console for a single aircraft system

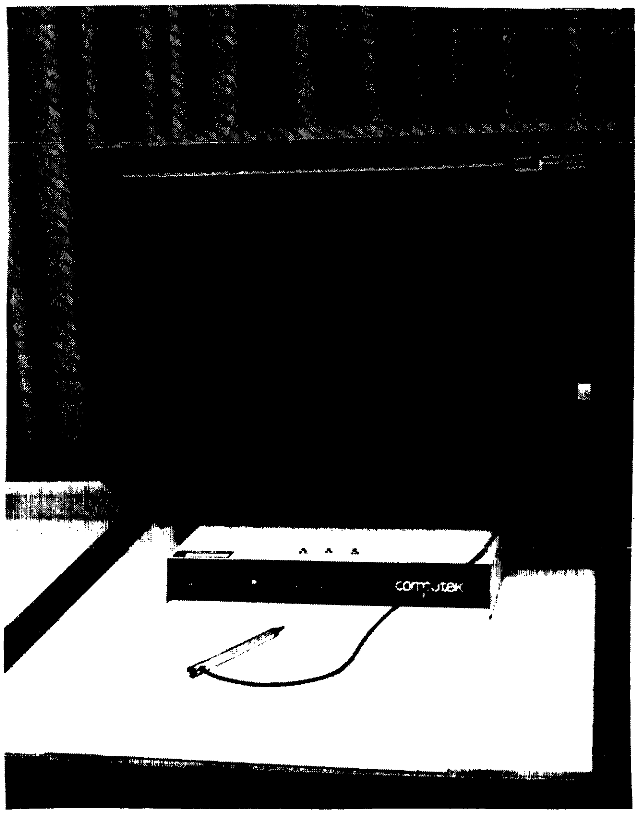


Figure 4b—Interactive computer graphics console for a six aircraft system

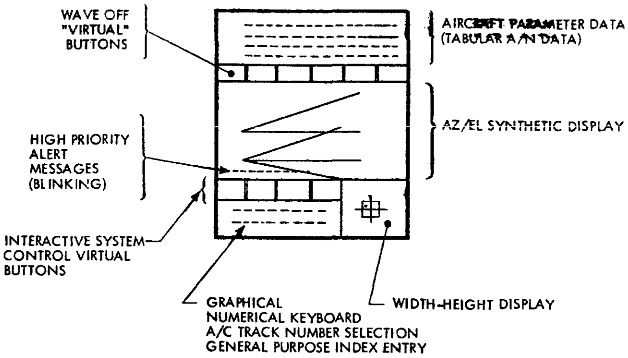

WAVE OFF "VIRTUAL" BUTTONS
HIGH PRIORITY ALERT MESSAGES (BLINKING)
INTERACTIVE SYSTEM CONTROL VIRTUAL BUTTONS
AIRCRAFT PARAMETER DATA (TABULAR A/N DATA)
AZ/EL SYNTHETIC DISPLAY
GRAPHICAL NUMERICAL KEYBOARD
A/C TRACK NUMBER SELECTION
GENERAL PURPOSE INDEX ENTRY
WIDTH-HEIGHT DISPLAY

Figure 5—Display format areas

format (21 inches is sufficient). The use of a round CRT compromises the optimum positioning of the tabular parameter, alert messages, status data, and virtual control buttons. Air traffic control displays which use round CRT's often place short tabular preview lists in portions of the situation surveillance area—and thereby obscure and mask possibly important data.

CODING MODALITIES

A number of coding approaches are being utilized in the MATCALS display format and interaction sequence. The modalities of color, intensity, line type (dot-dash), shape, blinking, and sound have been used to increase the range of coding dimensions, simplify the display format, and attract attention to specific information on the screen (Figure 6).

Burdick[2] has published the number of identifiable coding levels associated with these modalities:

Modality	*Coding Levels*
Color	3-10
Intensity	2-4
Shape (abstract)	8-16
Flicker (Blinking)	2-4
Line Type (dot, dash)	3-4

These modalities allow additional data to be displayed and absorbed by each operator so as to minimize the number of consoles and operators.

Color

The most intriguing modality, color is provided by the use of a slave beam penetration tube display, connected to the IMLAC. Assignment of display entities to the displayed four colors is as follows:

GREEN	*All background information,* parameter tables, time, wind, axes and labels, status, and control pushbutton menus.
YELLOW	*Aircraft and higher priority entities,* wave-off buttons (turns red when confirm is required), acquisition gate.
ORANGE	*Entities higher in importance than above,* aircraft symbols and tracking alphanumeric data blocks, glideslope vector.
RED	*Alert or Danger Indicators,* wave-off button confirmation, alert messages, symbols of aircraft in unsafe situations.

	Color Slave Monitor	Color Slave Monitor & Imlac Monochrome Display		
Item	Color	Intensity	Line Type	Blink/Flicker
Background	GRN			
A/C Parameter Table		Med		
Axes		Low		
Input Menus		Med		
System Parameter Readouts		Low		
Wave Off Buttons	YEL	Med		
Glide Slope	ORN			
AZ-EL Glide Slope		Low		
Aircraft		Med		
Acquisition Gate	YEL	Med	Dash	
Alerts	RED	High		High/Low
Wave Off Confirm Button	RED	High		

Figure 6—Coding modalities

It is recognized that color increases the effectiveness and information content of a display. Search time, selection accuracy, discrimination and counting of classes is minimized by the use of color coding. Color is also used in MATCALS to direct attention to portions of the displayed information base. The increased operator speed and accuracy due to the increased information content, realism and recognition is mandatory for MATCALS.

Intensity

Three intensity levels are being used:

HIGH	Alerts, wave-off confirm (used to draw attention)
LOW	Axes, and other background information (lowers the tendency of the display to be cluttered by low priority data)
NORMAL	all other data

Shape

The aircraft symbols are encoded as six uniquely shaped symbols (triangle, square, circle, half circle, diamond, and inverted triangle). These symbols are displayed on the Az-El surveillance area portion to indicate relative position, and displayed on the wave-off button and parameter list entry rows to allow rapid correlation and association of information by the operator.

Flicker

A combination of color, sound, and flicker was used to present alert messages. Blinking of messages annoys the operator. Red, while attention gathering, forces the operator to divert his eyes even after the message has been sensed. Inspired by Kubrick,[3] the display of a new alert message is initially red with a rapid flicker accompanied by a low pitch buzz sound. It then automatically changes to a silent, steady state red. When the operator first acknowledges the message, the color changes to a less annoying green. Subsequent acknowledgment deletes the message from view. Acknowledgment is accomplished by merely pointing to the message with the tablet pen.

VIRTUAL BUTTON CONCEPT

The graphical equivalent of hardware control buttons is presented on the face of the display. The operator may point to these "virtual" control buttons and the corresponding control function will be executed through the computer software.

For the MATCALS Display Console, a "Rand" X-Y data input tablet is used as the primary input device. The operator holds a stylus over a tablet surface horizontally mounted below and in front of the display CRT. This stylus can be randomly pointed to any position on the tablet surface. The computer system may read the X-Y position of the stylus over the pad (even when the operator holds the stylus some distance above the surface) and generates a cursor symbol (a small cross) on the face of the display. As the stylus is moved, the cursor on the display tracks the operator's hand-held stylus position on the pad as a feedback mechanism. A micro switch in the tip of the stylus is activated when the pen is pressed to the surface of the tablet. Therefore a button "push" is executed by the operator moving the pen to position his cursor over the "virtual" button he desires to push, and depressing the stylus. The operation is analogous to "pushing" a hardware button. This input strategy is very natural and the operator becomes immediately accustomed to its use.

HIERARCHICAL ORGANIZATION OF COMMANDS

The MATCALS Display organizes the operator's data input and system controlling actions into cohesive and logical multileveled groups. Each group consists of a number of menus of displayed virtual control buttons.

The hierarchical ordering of button menus allows the manipulation of commands and information requests. The command groups are ordered into a tree structure of usage and inter-relations. The user traverses branches of the tree (commands) as he progresses through the structure. This model attempts to describe the control relationships identical to the user's conceptual understanding and to guide him through his interaction with the control structure. Normally only classes of important control are displayed. When the user selects a class of commands, the detailed control types of sub-menu commands appears. When the user completes his interaction and use of these commands, he may return back to the higher levels of control types and command classes. The lower level is then automatically removed from the screen. We therefore have dynamic display and removal of command pushbuttons depending upon their syntactic requirement during an entry sequence. Extraneous and unnecessary controls are not continually displayed. A particular button menu is displayed only when its requirement is logically necessary. Any tendency toward a complex, cluttered, and confusing control panel presentation is avoided. The user can always quickly return to a higher command level even before making an actual entry at that level by pushing a "RETURN" button (which is always present). He may also directly return to the top level of the command class menu of virtual buttons when desired. All controller interactions with the system are checked for errors and provide feedback as to the entered quantity or function.

Interaction description

Figure 7 shows a primitive controlling initialization sequence to define an impending landing sequence. When he pushes the button "Select Landing Sequence," a new set of buttons appears. The operator may manually define the glideslope (and be taken down another branch where he enters a numerical value), enter the radar acquisition position, or (as is shown) select the desired Landing Mode. When this is pushed, the menu list allows him to select modes I, II, or III. As there is no computer communication with the landing aircraft in mode III, there is no need to enter a data link address and he is returned to level B and allowed to continue his data entry. If I or II is selected, he is allowed (at D) to enter an octal data link address. If he wishes to do so, a graphical numerical keyboard appears. In sequence, he may point to the desired keyboard buttons and the entered numerical value is accumulated. When he is satisfied with the value, he enters it by pushing the return button.

If no entries are made at a particular level, previously defined default values are assumed and used by the system.

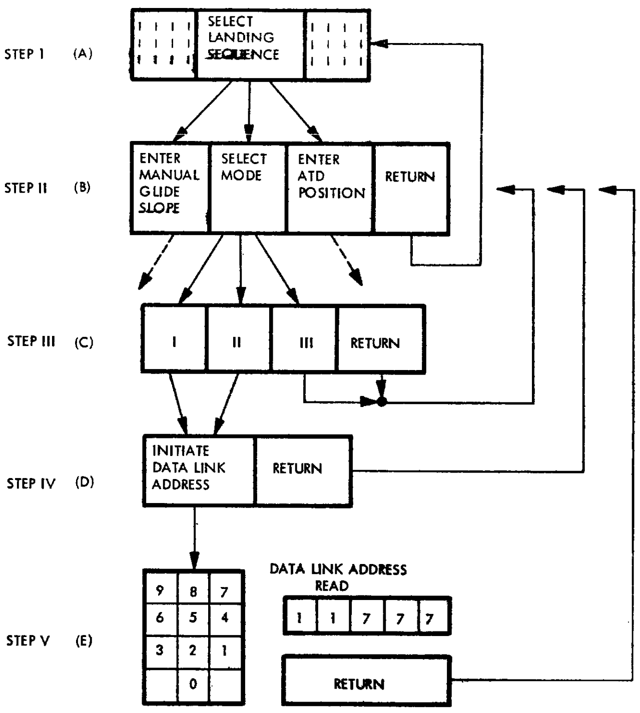


Figure 7—Hierarchical command selection sequence

This approach to computer input takes full advantage of the graphic display and of the human eye's ability to scan rapidly a single menu of commands. The entry rate is higher than by using a multi-function keyboard or array of hardware pushbuttons. The slowness associated with these hardware devices is due to the finite search time required to find the desired button and diversion time of the operator's eyes away from the primary display surface. The user does not have to recall a specific input control response (as with a keyboard) as all allowable responses are displayed. All displayed system status information necessary to make a decision is adjacent to the selection area.

Two different approaches to the ordering of levels must be mentioned. Generally, for initialization of a number of parameters a sequential multileveled ordering is optimum (a tall tree). Each group has a few selection buttons in each level menu. This forces the user to step through all the control initialization options to enter either the required data or to acknowledge that the system default value may be assumed (when he elects to go on in the hierarchy sequence without a data entry). This insures that an operator decision was made on all critical system parameters.

However, for modification of previously entered data, the scheme above would be frustrating to the user if he had to go through all parameters to get to the one he wished to modify. So instead, a shallower short tree of fewer levels is used for this case. However, each level has many buttons in each menu to allow the selection of the entity to be modified in a parallel fashion from a large list.

MATCALS utilizes two distinct multi-level trees:

a. Wave-Off Initiation Buttons
b. System Control and Parameter Input Buttons

The entire tree data structure is shown in Figure 8.

The System Control and Parameter Input Tree is the most complex example of hierarchical interaction with up to 6 levels of menus. Typically, the operator is lower than the third level only for brief times. The controls are grouped into six distinct classes:

a. Landing Channel Definition (aircraft initialization)
b. Landing Procedure Initialization
c. Landing Mode Selection
d. Radar Track Acquisition
e. Systems Controls (data recording and simulation)
f. Display Controls (range scale and other display format selection)

Within these classes are the levels of sets of virtual button menus that control the system or are used to enter system data. There are 2-6 buttons in each menu set. At many stages of data entry, general purpose menus are presented in the general scratch pad area to perform:

a. The entry of numerical values (using a virtual graphical keyboard)
b. The selection of an aircraft track number
c. The selection of a general purpose index

These general purpose menus may be thought of as interaction subroutines which may be used in any place in the control/data entry hierarchy. These graphical interaction subroutines appear only when required.

The use of hierarchically organized dynamic command panel menus in the MATCALS display system provides:

a. insured accessibility to information—fast update, reduced delay time, and improved spatial accessibility (no longer the need to search for that one correct button in a haphazard fashion).
b. increased reliability from the use of software to implement these complex control relations instead of the multi-wire/hardware switch approach.
c. increased flexibility—easier to adapt to required changes in types of controls.
d. increased utilization and efficiency of the control actions—improved vigilance of the observer—increased data input rate for untrained operators.
e. Commands may be tailored to the system understanding and educational level of the user/opera-

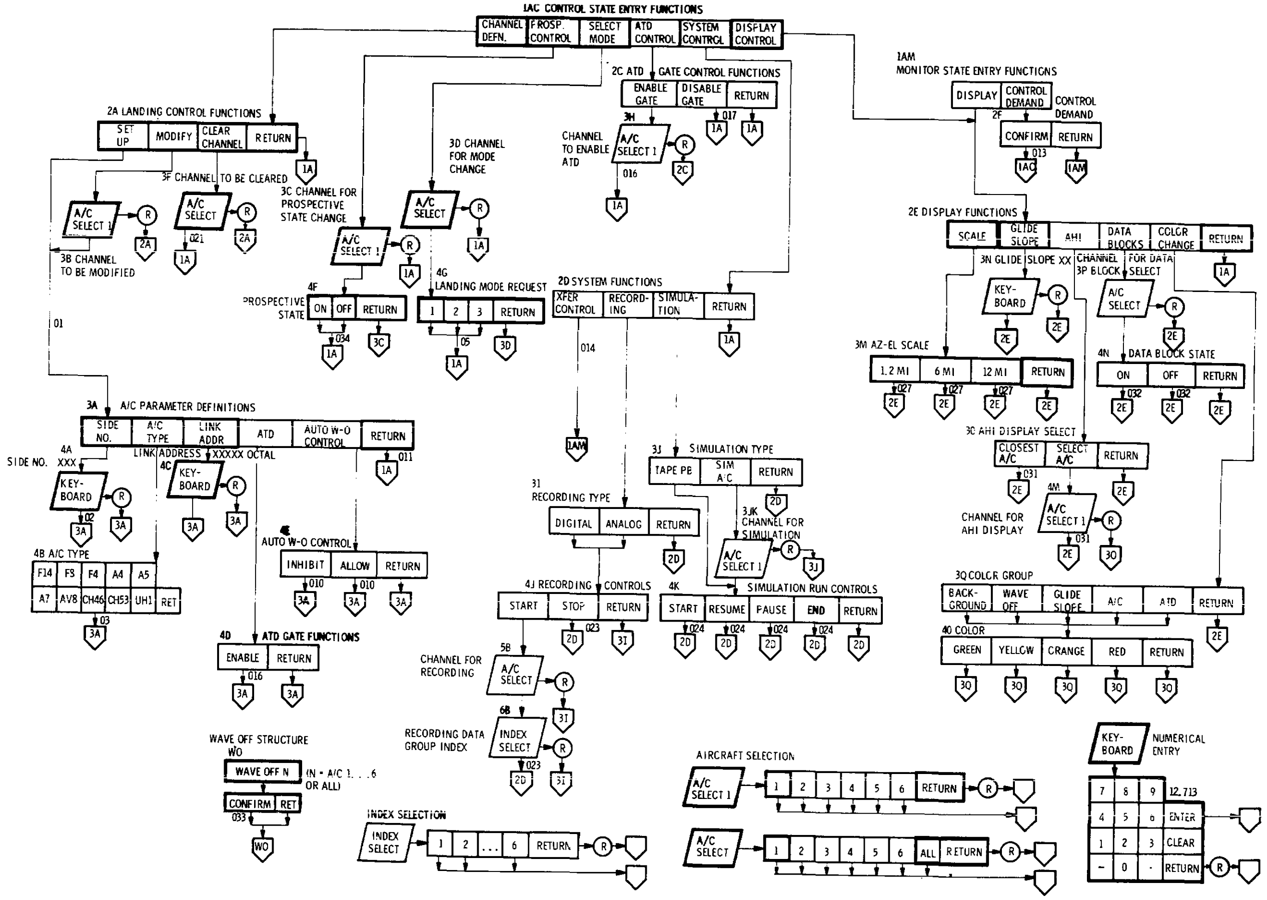


Figure 8—MATCALS interaction tree

tor. The commands are organized to allow the operator to teach himself the man/machine interaction.

Human factors considerations

Some quantitative research has been made on hierarchical dynamic controls using displays. Uber[4] has published that a maximal response time to this type of request is 0.5 seconds, which is one of the reasons for the high level of intelligence utilized in the MATCALS local display processor. Uber has also analyzed the optimal number of menu selection buttons per level. With a typical scan time of 0.5 seconds per entry, and selection time of 2 seconds per entry, the menu length is 7.7 entries. Experimental and trained users can have practical menu lengths of 10 to 40. Because of the possible background of the MATCALS user, a maximum menu length of 6 buttons has been selected.

DISPLAY FORMAT

Introduction

The display is divided into the following general areas: (Figure 5)

—Az-El Display
—Aircraft Parameter Table
—Wave-off Buttons
—Alert Message Area
—Interactive Control Buttons
—Width-Height Display

The Az-El area is the primary observation area. Glideslope angle and centerline, acquisition gate positioning, aircraft positions (6), and associated tracking data blocks are graphically represented. Three scales are selectable. Wind and time of day information is also presented. Aircraft positions on the display are rapidly updated to show relative motion of the aircraft.

The aircraft parameter area is a tabular alphanumeric listing of detailed information associated with each aircraft. Its use is considered secondary to the Az-El area, and more of a reference to less often needed data. The necessity for each type of data to be displayed for each aircraft will be evaluated during actual landings.

Wave-off buttons are continually displayed so they may be immediately accessible to the operator. Alert messages are presented in the lower corner of the Az-El area where they are immediately noticed.

The menu of hierarchical interactive virtual pushbuttons is presented on a row adjacent to the Az-El area for easy reference.

A Width-Height display (WHI) is presented at the right of the lower portion of the display. It is of fixed scale, and only one aircraft is shown. The WHI is primarily used for only the terminal portions of a landing sequence near touchdown.

Detailed description

A detailed description of the major display format areas as shown in Figure 9 is given in the following paragraphs.

Aircraft parameter table

Each aircraft is allocated one line in the aircraft parameter table located across the top of the screen. Each line of data is identified by the aircraft index and the associated symbol to be plotted on the Az-El and WHI displays. The aircraft parameter table contains alphanumeric data associated with each aircraft, including static parameters describing the particular aircraft and dynamic values related to the aircraft's position and movement. The static quantities remain fixed throughout a track while the dynamically varying data is updated every half second. Static quantities are side number of the aircraft, aircraft type (A7, F4, etc.), communication channel address, landing mode (I, II or III), and glideslope angle.

Dynamic quantities displayed alphanumerically are pitch and bank angle commands being transmitted to the aircraft (Mode I), sink rate (fast/sec), speed (knots), range (tenths of miles), height (feet), height error from the desired glideslope (feet), azimuth position (feet) and time-to-touchdown (in seconds). Corresponding parameters for each aircraft are arranged in columns and are appropriately labeled.

Wave off area

A row of wave off initiation buttons is placed immediately below the parameter table. This position puts the wave off area in close proximity to both the parameter table and the Az-El portion of the display. The wave off area is maintained separate from other interaction functions to allow continual and immediate availability to the operator. To speed up the wave off process, each aircraft is provided with a separate button, identified by the correlated aircraft index and symbol. When a wave off button is selected the operator is then required either to confirm that a wave off is to be generated or cancel the wave off request. Confirmation of a wave off request is required to lessen the probability of accidental pushing of a wave off button.

Az-El display

The Az-El display is centrally located and occupies the major portion of the display area. Aircraft height versus range is presented in the upper half of the Az-El display area and lateral position versus range in the lower half.

Provision has been made for a selection of three scales. Twelve mile, six mile and 1.2 mile ranges are selectable with a corresponding change in the vertical and lateral scales. Logarithmic or expanded linear scales are desirable alternatives that will be considered for future implementation. Axes are labeled for ease in estimation of aircraft positional relationships and the touchdown position is offset from the edge of the display to allow tracking for a short distance after touchdown.

A unique identifying symbol is used to plot each aircraft's position. The symbol is also shown in the parameter table to correlate the tabular and graphical presentations. In addition, a tracking alphanumeric data block is displayed adjacent to each aircraft position, connected to the aircraft symbol by a short leader. The aircraft index, side number, landing mode, simulated target tag, range, velocity and vertical and lateral glideslope errors can be read directly from Az-El display without consulting the parameter tables.

The time-of-day, wind speed and heading, displayed glideslope angle, and runway heading are listed in unused portions of the Az-El display area. The time is updated every second and is an indication to the operator that the control computer is active. High priority alert and warning messages appear in the lower left corner of the azimuth display.

Interaction and monitor area

The lower portion of the screen, approximately one fourth of the total area, contains the interaction functions. All operator/system interactions except wave off initiation and positioning the acquisition gate directly on the Az-El display are accomplished in this part of the display.

The function selection menus consist of several sensitive buttons that can be selected by positioning the tablet pen within the button boundaries. The display processor reads the pen position and determines what action is to be taken. A message accom-

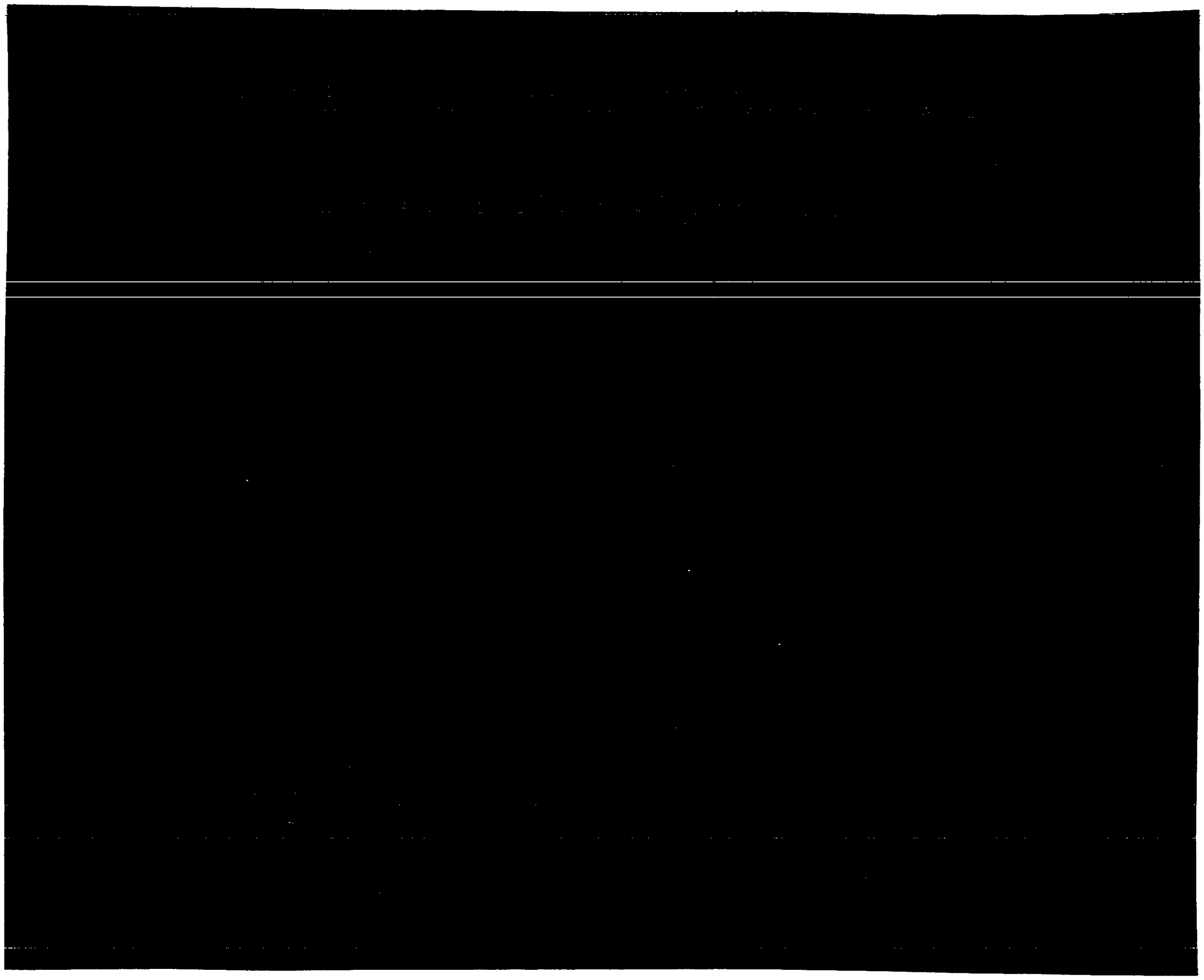


Figure 9—Display format

panies each menu to help direct and cue the operator's actions. Each button is labeled indicating its associated function.

When input of numerical data is required, a ten digit numeric keyboard appears in the interaction area with additional keys for minus sign and decimal point. The data is entered by sequentially pushing the appropriate buttons with the tablet pen. The accumulated digits are displayed as they are entered by the operator. Buttons are provided to allow the operator to clear the accumulator and begin again, enter the value if all digits are correct, or return with no data entry being made. When the numeric keyboard is no longer necessary in a data entry sequence, it is removed from the screen.

The rightmost lower portion of the display is used for the Width-Height Indicator (WHI) display to be used in conjunction with the normal Az-El display. Vertical and lateral glideslope errors can be plotted for any one of the aircraft shown on the Az-El display. The WHI has a higher resolution scale (±200 ft.) than the Az-El display and is used to determine more accurately the aircraft position close to touchdown. The system automatically selects the aircraft closest to touchdown for display unless this selection is overridden by the operator. The symbol correlated with the index number for the aircraft is used to plot the aircraft position on the WHI.

MULTILEVEL SOFTWARE DEVELOPMENT

Implementation of the display software was conducted in a top-down manner using the chief programmer team management philosophy and the guidelines of structured programming.[5] The system was developed in evolutionary stages with the first stage testing the high risk area of intercomputer communications and the central computer's operating system. In the first stage of the display program, skeleton modules ran under a primitive executive. As the implementation progressed, increasingly complex functional aspects of each module were developed.

The most important benefits of this approach were the early visibility of a working program, early assessment of the hierarchical interaction concept, simplified testing and integration with the risk reduction of integration problems, and elimination of the "90% complete" syndrome. Code review and the use of a program librarian were not significantly successful. The program team did achieve the ability to review each other's system designs and description with freedom from insulting each other's egos.

SUMMARY

The approach taken in this display design has led to an operational system which meets the objectives set forth in the introduction. Operators have complete control of the All-Weather Automatic Landing System from a single display console which presents an uncluttered view of the landing situation. The system is easily learned, operated, and requires no knowledge of computer systems. The hierarchical interaction strategy for data entry and system control has been proven to be both accurate and fast. Measurements taken on operator performance have determined that a 0.7 second per button selection rate can be maintained with the hierarchical interaction technique. The operator entry of all initialization data required for an automatic landing sequence (some seventeen virtual button selections) is accomplished in only twelve seconds. The operator is aware at all times of the permissible controls which are active and what possible actions he may take. Favorable comments have been received from a number of experienced landing system controllers, who feel the interaction and display concepts will increase operational safety.

ACKNOWLEDGMENTS

This effort was supported under Contract N00228-75-C-4544 with Mr. John Mulhern of the Naval Electronic Systems Engineering Center, Vallejo, providing many helpful technical suggestions. Discussion with Mr. Richard Wilz, NAVELEXSYSCMD, Washington, and Mr. Ken Potyen, NESTED, Patuxent River, Maryland, were also valuable during the development of the display system.

Implementation of the computer programs was accomplished by George Firkins, Susan Rubin, Lowell McMahan, Leo Hoffman and Al Crown. Project support was provided by R. L. Goodman and R. L. Johnson.

REFERENCES

1. Wilz, R. R. and A. Warmack, "MATCALS: Expansion of Capability for Expeditionary Airfields," *20th Guidance and Control Panel Symposium*, North Atlantic Treaty Organization, Cambridge, Mass., 20 May 1975.
2. Burdick, D. C., et al. (Naval Research Lab) *Color Cathode Ray Tube Displays in Combat Information Centers*, U.S. Naval Research Laboratory, Washington, D.C., Report NRL-6348, October 1965.
3. Kubrick, S., *2001: A Space Odyssey*, MGM, 1968.
4. Uber, G. T., Williams & Hisey, *The Organization and Formatting of Hierarchical Displays for the On-Line Input of Data.* FJCC, 1968, pp. 219-226.
5. Baker, F. T., "Chief Programmer Team Management of Production Programming," *IBM Systems Journal*, January 1972, pp. 56-73.

Libraries and the implications of computer technology

by MURRAY TUROFF and MARION SPECTOR
New Jersey Institute of Technology
Newark, New Jersey

ABSTRACT

This paper examines the potential impact of computer and information technology on the role and mission of the library. In an indirect, but no less significant sense it also considers the impact of the library on the technology. The specific concerns focus on the library as a mechanism for making computer and information technology directly available to the public. The paper also reviews a pilot project being conducted by the New Jersey Institute of Technology involving the Newark and Millburn public libraries in New Jersey.

INTRODUCTION

As of January, 1976 a number of powerful calculators have been installed in the Newark and Millburn Public Libraries as well as the library of the New Jersey Institute of Technology. The Newark Public Library serves an urban community which reflects the diversity of population and the multiplicity of problems facing most major urban areas in the country. In addition, the Newark library is the official state regional library for the northern New Jersey area and as such offers a number of unique collections pertinent to this mission. The Millburn Public Library reflects an upper middle-class suburban community. A majority of the high school students go to college and the working adult population is largely businessmen or professionals. The NJIT library services a scientific and technically oriented university community of about 5,000 students. It is largely a commuter-oriented institution for northern New Jersey and the Metropolitan New York area. A significant proportion of the students are from blue collar families and a very active program for the disadvantaged has attracted a good number of these students into the engineering and technical programs.

None of the above mentioned libraries have had any previous experience with computer or information technology. They have not utilized the marvels of computerized reference services, data bases, automated cataloging, etc. Although a calculator might seem a limited aspect of computer technology, in the environments under consideration the introduction of these devices represents a major shift of perspective along with a rethinking of roles, missions, and objectives.

Principally, this equipment is for the explicit use of patrons for any purpose they wish. The primary goal, as stated in public flyer, is making directly available to the public via the public library various computer and information services. This is not to say librarians do not use the machines. In fact, the machines installed are in the five hundred dollar range and considerably more convenient then the machines that existed solely for internal use.

No project of this type exists because of any one rationale. The members of the project team, users, librarians, sponsors, manufacturers may very well perceive this project differently. The implications are perhaps best understood by viewing a number of alternative rationales that are or could be representative of the parties involved.

SOCIAL CONCERN

Approximately two years ago one of the members of the project team was in a Washington, D.C. department store and happened to observe a salesman driving away some young children who wanted to play with the calculators on display. Evidently he did not perceive them as likely customers. To those of us who have envisioned computer technology as accessible to the public such an occurrence is personally disturbing. However, disturbances of this type have a way of generating reflection.

It seems that for many years the view of many professionals was that the all-encompassing computer 'utility', because of economics of scale, would be the mechanism for bringing computer power to the people. While hardware costs have demonstrated their economics of scale, software and communications will remain a bottleneck for quite sometime. Even while this orientation toward the grand large scale system continues for a vast number of prophets in the field, it has turned out that for the people 'it is the little things that count'.* The calculator is the first major item of

* Attributable by the authors to Richard Wilcox.

computer and information technology to fall in the hands of the public on any wide-scale basis.

Perhaps the events arising out of the selection process that occurs in a competitive market place are hinting that it is not large scale but the small scale we should be looking at for delivery mechanisms. There have been in the past few years a growing number of such small scale-efforts—Resource One in the San Francisco area for example.[1] The majority of endeavors appear to be the volunteer efforts of concerned professionals and community groups. Unfortunately, because of the nature of these activities many of them lack documentation and very little evaluation is accomplished. While many of the individuals behind some of these programs might scoff, we feel they have suffered from a lack of institutionalization. What is sacrificed, of course, is the opportunity for permanency and wide scale transferability.

With these considerations in mind our choice was to start at the bottom, utilizing a variety of calculators with performance and associated price above the level the average person could afford. It was also a conscious choice to institutionalize the effort—but why the library? In one real sense it is like the explanation of why people climb mountains: they are there. The library is an institution that is available to the public, its personnel are familiar with serving the public, and it is relatively neutral with respect to political, social, ethnic, and organizational polarizations. From the point-of-view of a person who is interested in delivering computer technology to the public the library is the convenient place to do it.

Why one wants to put advanced technology in the hands of the public is probably a question each individual should answer for himself. However, one view is that it may be increasingly difficult to exercise the privileges of an intelligent citizenry in a democratic society without an understanding of the capabilities and limitations of the technology which is beginning to monitor, regulate, and perhaps control aspects of that society.[2] The key issue seems to be: "Will the utilization of this technology by society be such that each citizen must have a right of access and availability in order to function as a part of the society?" Suppose today we took a group within our society and denied them use of the telephone? One could easily list a set of severe consequences for such a group.

LIBRARY MISSIONS

The average person probably equates the usual library with vast arrays of printed material such as books or other physical images like films, records, etc. The more appropriate view for our purpose is to consider the library as an institution for allowing people to utilize information. Utilization implies not only storage and retrieval, but creation, organization, and manipulation as well. The use of the technology to allow patrons to be able to directly perform these latter operations implies a host of information services that have not previously been possible:

- Allow the library to support transient information needs of its user community.

  Individuals able to develop their own personal data files and text files on electronic storage media such as floppy disks—to manipulate, update, and edit these as need be.

- Provide mechanisms for patrons to exchange information.

  Essentially the community electronic bulletin boards and the computerized conferencing[3] with discussions going on of any topics of interest to the users of the library.

- Establish the Library as a Learning Resource Center.

  A place where individuals of ages 5 to 85 can go to take advantage of the educational options offered by the technology.

The possibilities stretch from rather straightforward electronic bulletin boards, text processing, discussion systems and CAI implemental on microprocessor oriented intelligent terminals with floppy disks or digital cassettes to more future oriented visions as those expressed eloquently by such individuals as Kagan and Nelson.[4]

In essence the key ingredient is the concept of easily dealing with "variable" information. Information subject to change and modification on a frequent but unpredictable basis.

The choice of the library and its institutional characteristics has led us to another premise in our design of this effort. Most libraries have been reasonably successful in the performance of their mission because the services they provide have no short term dependency on other organizations in the delivery process. Short of the dramatic act of closing a library, most sponsoring institutions do not (at least not yet) instruct libraries to sell books to raise money, whereas colleges can be told to cut down enrollments, offer less courses etc. With this in mind a choice has been made to focus on equipment which is self-sufficient, and not dependent on outside computer power. This is not to say that if we install at some point, an intelligent terminal or word processor, it will not have the dual-use capability of being able to talk to a computer. In fact, any reasonably sized organization buying word processing equipment that cannot be upgraded for this, has not looked to the immediate future.

EDUCATIONAL ASPECTS

Although there are public schools which offer experience with computer technology, they present a distorted perspective of the extent to which technology is actually being taught to the public. There are many outstanding programs in largely upper middle-class communities or in communities in proximity to a university. These efforts represent only a fraction of the pupil population. For example, in the near future, it is unlikely the Newark New Jersey school system would be able to replicate the six thousand dollars worth of calculator equipment available in the Newark Public Library within each of its schools. Even if it did, the equipment would not then necessarily be available to other specialized educational programs without added staff costs to open school facilities after school-hours.

In the case of a public library, one must consider a diversity of user needs for computer technology that must be met. They run the gamut from advanced research to recreation. The objective of expanding computer technology depends, as well, on the nature of the user population. In the case of this study, equipment for the Newark Public Library (a lower-income area library) may not be appropriate for Millburn (an upper-middle income area library).[5] Similarly, one would allow for more sophisticated equipment to be included in the library at NJIT, a technical college library. The sample of this study was especially chosen to represent two socio-economic communities as well as a technical college community. It will thus be possible to evaluate differences between the groups from the point-of-view of the appropriateness of the new equipment. The principal question to be answered is whether this advanced media equipment represents any improvement over past services.

The library has traditionally been the place where individuals go to learn on their own. It serves as an important place where users can pursue their own information needs. Librarians continue their efforts to encourage users to consider learning as a life-long process. Therefore, an objective of this study is to investigate if the library cannot serve as an improved educational agency. In addition to introducing computer technology to the public, this service of the library could be used efficiently in the educational process.[6]

It is clear that the adult population particularly from thirty on up has not been adequately serviced by traditional educational programs. Many adults no longer have the opportunity to attend school. For those that do, there are many problems associated with returning to college or even to graduate school. Imagine a practicing engineer in a class of engineering students becoming aware that he knows less than some of the young students! Technical areas of instruction, particularly where updating is concerned, are ideally suited to CAI (Computer Assisted Instructional Systems) made available on a self-service basis in libraries. One of the earliest lessons to be learned by those involved in designing operational Management Information Systems is if you want senior managers to use these systems and do not have enough terminals for every office, you put it on wheels so the manager can take it into his office and not publicly demonstrate to younger employees the mistakes he is making in learning the system.

NJIT has developed a number of specialized educational programs to provide opportunities for disadvantaged students to enter technical fields. These programs have partially blurred the distinctions between course materials offered in the high school and in special first year programs at the university. They have entailed tutoring efforts, summer sessions and weekend classes for high school students, special classes for high school teachers, as well as the development of instructional materials for the use of high school teachers.

Educational institutions involved in the programs for the disadvantaged, could pool their resources, and invest in Learning Resource Centers located in libraries. From an educational standpoint the availability of computer technology able to support drill and practice as well as the updating of basic knowledge, would greatly alleviate the coordination difficulties that result from offering remedial services in public schools, universities, and adult programs.

This pilot project will emphasize guidance and instruction in the use of computer technology to teachers. Their willingness to use the media made available is essential to the success of this program. Their perception of the role of educational technology is of the utmost importance.[7] It will determine the length of time they are willing to spend developing specific educational lesson plans for the equipment. They will be encouraged to take advantage of the calculators to design stimulating assignments for children. In mathematics, children could easily handle more challenging problems such as computing a growth curve to fit the daily growth of a classroom avocado plant. This computation would normally be too laborious without the use of a calculator.

From the point-of-view of educational theory, there appears to be a shift from concern with teaching to a concern with learning. Whereas, teaching provides a framework for learning, teacher-centered learning does not develop independence in the learner. Teaching is done to the individual, not by the individual.

Educational technology may be an effective technique for students to acquire skills independent of the teacher.

A question to be explored is what types of individualized learning experiences students can handle in different grade levels as a result of educational technology? Also, computer technology is especially important for providing work at varying rates for students at the extremes (faster or slower) than the average classroom. It is expected the library could serve as an adjunct to the teacher for these students. For example, this study will use drill and practice techniques as one effort for those needing remedial mathematics. Applying the psychology of learning to practical teaching problems, the student is presented with individualized instruction in arithmetic.[8] Actively involved with his own performance of arithmetic problems, the correct answers via computer print-out serve as an incentive in the learning of repetitive tasks. It is no longer necessary to wait to have a test scored. As far as assisting the classroom, drill takes up considerable time. Drill and practice take time away from teacher-contact with students that could be used in better ways.

Generally schools and libraries have co-existed as separate units. An objective of this study is to determine professional (librarians and teachers) roles in the use of educational technology. It is possible that teachers will be more receptive to educational technology when it is in the library. They may not sense the equipment as a threat to the teaching force.

Many teachers have been dubious regarding the importance of the new material and have not taken it seriously. This study will utilize survey instruments as well as interviews to evaluate the attitudes of teachers and librarians.[9–10] They will also be working with the media specialist in the selection and utilization of materials. Results from questionnaires will serve as an up-to-date monitoring system to determine which tools are serving advantageously and which are falling short of expected outcomes. The results will be used as a basis for future priorities, deciding the long-range goals of this program.

PROFESSIONAL CONSIDERATIONS

The concept of technology assessment has emerged in recent years as an issue of primary concern to society.[11] Most assessment studies have been executed as paper and pencil efforts. It has yet to be realized that there exists a need for assessment by experimentation. From one view this effort could be considered a 'live' technology assessment. A major goal in this assessment effort is to gain insight into the unexpected consequences of technology. We expect to observe results we could never have predicted. A few unanswered questions in this area are:

(1) For what purposes will the technology be used by the public and what are the benefits or limitations?

(2) Will the library schools have to train educational librarians (will SLA add a new component to its membership)?

(3) Is the technology designed to really meet the requirements of this mission?

(4) While we have in Information and Computer Science had some success in designing to deliver "Information," do we know how to design to deliver "Information Technology"?

Even at the calculator level the variability in human interface for machines of similar capabilities is quite significant. In terms of advanced machines, the layout of the keyboard and the associated instruction manuals appear to be either for the sophisticated user or the individual who *must* learn to use the machine. The developments in this area have been so rapid that few of the companies appear to have made any evaluation efforts in terms of market assessment or product development. This may change as the market saturates.

Another aspect of the assessment issue is the observation that Congress seems to be dragging the National Science Foundation, kicking and screaming, towards a program called 'Science for the Citizen'. No one at this point seems to be sure what that program might encompass. One would hope the delivery of advanced technology would be a key aspect. It is quite clear that organizations such as the National Association for the Public Assessment of Technology would tend to support a view that the public should have a right of access to information and information technology if it is to have an equal voice with those institutions that can afford these benefits.

Technology has moved so rapidly it always appears that implications for the training of professionals in these fields takes place as a reactive process as opposed to an anticipatory one. For example, the vast majority of development and design activity in the professional community represented by ASIS has dealt with the technical means of retrieving data for the user but not the processes of allowing users to create, store, manipulate and update data.

We can foresee that as a result of programs designed to improve the availability of technology for the public, there will be a further blurring of the divisions between Computer Science, Information Science, Library Science and the Social Sciences related to educational processes.

ISSUES AND PROBLEMS

This paper has raised a number of issues and problems. Whether it be at the limited level of the calculators, or the more sophisticated equipment we hope to incorporate later, the following is a summary of issues that must be examined, evaluated and resolved before one can expect any wide scale implementation of programs of this type.

Community acceptance

There are a great many individuals with unimpeachable credentials whose current reaction to computer and information technology is to avoid it like the plague.

One can point to the existence of this antagonistic attitude represented even on our own faculty. Perhaps more serious a concern is the feeling that the technology is an influence in widening the gap between the disadvantaged and the rest of society. Apparently some community organizations have taken positions that they do not want calculators allowed in public schools because it will degrade the child's learning of basic math. Also, some experiments with the availability of CAI systems, have indicated that a child having difficulty in relating socially to other children may further withdraw.

This latter phenomenon is also behind some success stories with Management Information Systems. Given an organizational environment that has severe restrictions on human communication, an MIS system that is designed to be very reactive at a terminal, can easily become an unconscious surrogate for human communications—the computer becomes someone who will listen, obey and respond.

One key element in the library approach is the volunteer nature of the effort. The use of the calculators is a self-made choice by the individual. This may lessen possible friction involved if these were imposed directly into an educational program. We are trying to maximize the availability of material on games and puzzles which represent an unconscious form of learning, and the existence in the program of the 'Computer-Tutor' and the small 'Quiz Kid' means the teacher can encourage those children who need basic drill and practice to engage in such. The more advanced machines for either business or scientific use can only be mastered if the individual knows or attempts to learn the basic applied math concepts that are presumed in their design. Also we have undertaken to instruct those teachers who want to learn as to the types of lessons that could be assigned in math and science areas that would not be convenient without these machines—e.g., curve fitting to plant growth data. Since we are not set up to have a full-time educational staff available, the procedures encourage people to help one another to learn. Initial eyeballing of what is taking place appears to indicate that children are most often learning and doing on a machine in groups of two or three rather than one alone. Maybe our concept of a CAI workstation, with one child working alone at a terminal, bears re-examination.

CAI design

The CAI area can be characterized today by two diametrically opposite approaches: One is the large system with a great many hours of effort put into developing a single lesson by specialists trained in the intricacies of the system. This could be represented by the PLATO system and the TUTOR language at the University of Illinois. An alternative approach is the PILOT CAI language which represents ten simple commands that can be taught in about one hour and allow any teacher to prepare straightforward lessons to be used by students. Interestingly, the PILOT has been implemented on the DATAPOINT Intelligent Terminal, and can be implemented on any of the similar devices made by such companies as DEC, WANG, etc. The PILOT language can also be taught to sixth graders on up and utilized to allow students to design lessons for other students which represents another intriguing learning approach, particularly for the student designing the lesson.

Basically, these two approaches can be distinguished by: one, canned lessons prepared by specialists and two, tailored lessons modified or created by the individual teacher and the students. The latter is implementable on stand-alone devices costing today about twelve thousand dollars. One can infer from other parts of this paper that we feel this latter small scale approach belongs in public libraries as part of a Learning Resource Center concept.

Some other rather obvious issues we have already discussed are:

Institutional Cooperation—Ultimately the problems of cooperation of educational and library institutions have to be faced, not only at the funding and administrative level, but at the training level of professionals employed by these institutions as well as the research and evaluation efforts associated with these endeavors.

Recreation and Learning—It is really not clear that we understand these processes or their relationships and potentials within the context of the technology we have been describing. The little calculator has already raised issues about possible dis-benefits to the student without accurate evaluation to support any of a multitude of views. The calculator is only the tip of the iceberg with respect to the potential technology.

Equipment Design—Current equipment design is far from satisfactory for the purpose of general public use. This is even more true as we move up the ladder to intelligent terminal capabilities. Also, instructional material supplied by manufacturers is usually written with the assumption the user *has* to learn the machines. Security of the equipment is another aspect that has only received minor attention from some manufacturers.

Problems with Change—Both librarians and educators can suffer from the same psychological resistance as other adults reluctant to reorient or engage in updating their knowledge and rethinking basic goals of their endeavors. As a whole, we have been very fortunate with the individuals involved in this project. However it does take time and patience to bring about an understanding of an endeavor of this type as it is not established practice for libraries. Also, a number of libraries that have made available simple hand-held calculators, in the twenty dollar range, have had to drop the programs because of maintenance and loss problems. Most public libraries do not usually have the background to judge the machines required and to evaluate their capabilities. It has been our approach to provide the expertise to utilize advanced equipment that is not competitive with what the average person can afford. This then provides the library patron a service that is potentially attractive in terms of what he or she may need to accomplish.

REFERENCES

1. Colstad, K. and E. Lipkin, *Computers and Society*, Vol. 6, No. 4, Winter 1975.
2. Turoff, Murray, "View of the Future," *AFIPS Conference Proceedings*, Vol. 42, 1973 NCC.
3. Turoff, Murray, "The Future of Computerized Conferencing," *Futurist*, Vol. IX, No. 4, August 1975 and *The Delphi Method*, edited by Linstone and Turoff, Addison Wesley, Advanced Books, 1975.
4. Kagan, Claude and L. G. Scher, "The Home Reckoner," and Nelson, T. H., "A Conceptual Framework for Man-machine Everything," *AFIPS Conference Proceedings*, Vol. 42, 1973 NCC.
5. Gaver, Mary, *Effectiveness of Centralized Library Services in Elementary Schools*, Rutgers University Press, New Jersey, 1963.
6. Enright, B. J., *New Media and the Library in Education*, Linnet Books, Hamden, Connecticut, 1972.
7. Suppes, Patrick and Mona Morningstar, *CAI at Stanford 1966-68*, Academic Press, New York, 1972.
8. Suppes, Patrick, Max Jerman and Guy Groen, *Arithmetic Drills and Review on a Computer Based Teletype*, EIC ED 014 215, November 1965.
9. Darby, Charles, A. and Arthur Korotkin, *The Computer in Secondary Schools*, Praeger Publications, American Institute for Research, New York, 1972.
10. Goodlad, John, John F. O'Toole, Jr. and Louis L. Tyler, *Computers and Information Systems in Education*, Harcourt Brace World, New York, 1966.
11. Mitroff and Turoff, "Technological Forecasting and Assessment: Science and/or Mythology," *Journal of Technological Forecasting and Social Change*, Vol. 5, pp. 113-134, 1973.

APPENDIX—SCENARIOS

One way to place the goals of this program into concrete terms is to list some specific scenarios which show the various uses to which this technology located in the library can be put.

A high school student uses a calculator to do some calculations on physics homework.

A housewife uses a calculator to balance the family checking account and evaluate the impact on the planned budget.

A professional (engineer, accountant, etc.) comes to the library to utilize a more powerful calculator than he or she could normally afford.

Many teachers at local schools can restructure their homework assignments in the math and sciences to reflect the availability of calculators at the library.

A secretary and/or treasurer of a local club, organization, community group walks into the library carrying his data file on membership and finances in the form of a small digital storage device—e.g., digital cassette, minitape, floppy disk, etc. He or she goes and sits down at an intelligent terminal that has been set up as a text editing and composing device and performs the updating on recent changes in addresses, membership, dues, expenses, etc. He or she then prints out a summary and duplicates copies for the membership.

A student utilizes this same storage media and terminal to compose and edit term papers over a period of time.

A new form of book called the 'interactive book' has become very popular. With this book stored on a digital storage medium, the reader can play the role of one of the characters so that as he reads he can make choices at strategic moments in the plot which result in different plot resolutions.

Even the librarian can utilize the word processing equipment to prepare various reference lists with continuous updating capability.

A young couple walks into a library and asks for the digital cassette holding the advertisements of babysitters available in the neighborhood: a teenager walks in to add his name and description to the same cassette.

A teenager requests the cassette or floppy disc advertising part-time help available for cleaning, gardening, snow clearing, etc., so he can add his name and capabilities to it.

A citizen requests the cassette devoted to discussing a new and provocative book so he may review what other people have said and add some of his own comments.

A teenager requests the cassette of an anonymous discussion of dating problems by teenagers. An adult requests the cassette dealing with marital problems.

A group of local businessmen or procurement officers have a cassette on which they are exchanging notes on the performance of various supplies, equipment and suppliers.

A cassette is being used by people in the community to discuss their views on various local issues, say a rezoning item. A copy is sent once a week to the government group involved.

Various cassettes have been prepared and are maintained by various local government and community groups to provide information on services, volunteer work available or needed, etc.

A psychologist is utilizing cassettes in the library as a form of random group therapy sessions.

A social worker has organized and monitors a number of problem oriented conferences among people having similar difficulties.

A class discussion cassette has been established with a local teacher to augment the regular class hours and provide more opportunity for free discussion. This is a particularly popular facility for use in adult education courses.

A group of stamp collectors utilize a cassette to bid on stamp trades.

A group of professionals in a common area of endeavor utilize a cassette to exchange information on recent papers and findings in their field.

A teacher delivers to the librarian a floppy disk containing a CAI lesson she has prepared for use by her class.

The expanding role of on-line interactive searching

by VIVIAN S. SESSIONS
The City University of New York
New York, NY

ABSTRACT

On-line interactive searching represents the newest thrust in information retrieval. The combined technologies make available to librarians and other information professionals large volumes of bibliographic data which reside in computers as remote as thousands of miles from the terminals using them. Presently the individual data bases that furnish the examples of actual usage (ERIC, NTIS, Psych Abstracts, etc.), are accessed primarily via Lockheed, SDC, Medline, and the SUNY Bio-Medical Network. Although it is difficult to give a definitive statement on the proportionate economic contribution of this kind of information retrieval to the entire computer economy, estimates of numbers of characters of information on-line (25-50 billions), indicate a significant dollar involvement. It is in this framework that bibliographically-oriented members of the computer community urge their non-bibliographic colleagues to understand information retrieval and to heed its requirements for further development.

Of all the library and information center usages of computers, the one that concerns this paper is on-line interactive information retrieval. It is retrieval in the sense of finding references to documents relevant to a particular problem in data files that are comprised of bibliographic records. Not all such data files can be used either interactively and/or on line, but those that do not qualify on both counts are excluded from this discussion in order for us to zero in on the I.R. mode that is today most visible in terms of public discussion, in terms of growth, and in terms of challenges to the computing community.

A typical form of on-line interactive retrieval would be the situation in which an information professional uses a terminal with dial-up capabilities to log into, say, Lockheed Information Systems in California via a communications network. S/he indicates a particular data base (the Educational Resources Information Center (ERIC) is a good although over-used example), and then proceeds to process such a query as the one recently done at the Center for the Advancement of Library-Information Science at the City University of New York. This particular query was for references to materials on collective bargaining in higher education. The CRT quickly indicated that the particular way in which the query was asked would yield 367 references in the ERIC data base. The terminal operator asked to have a few displayed, and when all of them seemed on target to the requestor who was sitting by, a message was sent to have the entire number printed out on the high speed printer in California. In a few days the information arrived in New York.

As this was one of the less expensive data bases, and very little computer time was needed to develop a strategy, the total bill was about $40 for the combined use of the data base, the communications network, and the high speed printer—mostly the latter, with 367 citations @ 10¢ per citation printed. This cost excludes, of course, the original investment in the terminal and the time of the information scientist who served as the interface between the requestor and the system. Since the requestor was satisfied with 367 references, there was no attempt to query the file with alternative strategies, thereby keeping the connect time, the most important cost item, at a minimum.

Characteristic of the kinds of systems under discussion in this paper is the fact that the data bases are loaded on computers that are remote from the terminals accessing them—remote not just in the sense of being in a different part of the same building as the terminals, but remote as measured in hundreds or thousands of miles, and housed in different organizations. Even the creators of the data bases (The National Technical Information Service, American Psychological Association, to name a few), once they send their regularly scheduled up-dates to Lockheed or SDC, then access their own files over the same kinds of terminals and via the same kinds of communications networks as the users of their data bases. (Some specialized exceptions do not affect the general situations.)

Remote on-line information retrieval only became a publicly available resource in libraries and information centers in 1972, with a limited number of files. By 1974 Lockheed and SDC had developed their respective capabilities to the point that the Information Industry Association awarded them jointly the Product-of-the-Year award in 1975. In the year since the award they have made considerable advances in the terms of services and customers. Together these two private corporations now account for two-thirds of the kinds of bibliographic data now available for on-line remote searching in the U.S., the other third being provided by public systems, the files of the National Library of Medicine and those of the SUNY Bio-Medical Network. Although there are large overlaps in the data bases handled by these four systems, they can be considered as being independent contributors to the computer economy, for reasons that will be pointed out shortly.

The fact that some of the Lockheed and SDC data bases are government-created and some are private in origin, such as the Institute for Scientific Information (ISI), is of significance to the user only inasmuch as the privately created data bases are more expensive; methodologies for using them differ to the extent that every data base has its own characteristics as to coverage, access points, and hard copy aids to users. And all these are constantly changing.

In the decade before the remote on-line interactive developments, many librarians were already involved in information retrieval. If they worked for organizations with the resources necessary to mount their own bibliographic information systems, they could directly participate in the vanguard of their profession. However, these opportunities were limited to librarians in large corporations and government agencies; the public and university librarians were out of it except for those few places that had contact with information dissemination centers.

Now, however, direct participation in the information retrieval process is possible to anybody having access to a terminal with dial-up capabilities, access to a purchase order of a few thousand dollars, and access to some training (more is obviously preferable). Although many information professionals may be forced by institutional reasons or personal inclination to continue depending on intermediate search centers, the desire to be independently on-line to immense storehouses of bibliographic data is less a matter of wishful thinking and more a matter of persuading internal management of the desirability of such action. The advantages to the individual information professional for upgrading his/her skills aside, the advantages to the organization are enormous. The more its own professional information staff knows about information retrieval, the more that organization is going to profit from it.

"On-line" and "interactive" are usually stressed rather than the "remoteness" of the data bases in present-day discussions of information retrieval. All three are intertwined. It is the on-line capability that brings remote access to life, and it is the interactive characteristic that really separates on-line from batch searching. Without the interaction, information retrieval on-line offers little advantage in search strategy to the batch mode.

The interaction now possible on the standard on-line information retrieval system is not as conversational as the term implies, but has divided itself into systems—type interactions (such as the user being told that the host system is on line, or that the operator has made a syntactical error), and interaction wherein the user knows immediately (computer-time immediately) how many document references would be retrieved in the NTIS system by using the search terms being investigated. For instance, how many "hits" by using the search term "service industries", how many would be found by the term "innovation", and how many by the combination of the two in an AND strategy? It quickly developed in this search that the results would be insufficient unless "service industries" was broadened to include "services" in general in an OR relationship.

To illustrate another interactive feature in the very versatile NTIS file, the information professional looking for material on "information theory" might do well to use the EXPAND command, which would then put on the CRT all the descriptors in which "information theory" is only the first part of a multi-word descriptor. This could result in a decision to either broaden or narrow the search, the latter being done by asking only for "information theory, band widths", if that is more appropriate. The EXPAND capability is particularly important in systems that either have no printed thesauri, or in situations in which the user has no immediate access to the thesaurus.

One of the most powerful features of interaction has already been mentioned in connection with the ERIC search earlier: the ability to display a few citations to see if they are on target, thereby enabling the searcher to decide whether to continue modifying the strategy, or to work with what has been retrieved, and order a print-out. If the searcher is using a terminal with a printer attachment, or a printing terminal, the only decision left is whether the results are either sufficiently short or sufficiently urgent to have them printed out on the spot, or to leave them to the high-speed printer and the mail. These few examples of the effects of interaction on search strategy merely scratch the surface.

The intricacies already mentioned of searching interactively, and many other fine points extending the possibilities, are starting to be taught to the librarians or other information professionals in a variety of formats. This is quite different from library automation, which concerns itself with circulation control, catalogue card production (now handled in volume through the Ohio College Library Center) and related administrative tasks. While such uses of the computer contribute heavily both to the smooth functioning of the library and to the computer economy, it is information retrieval which responds to the expectations of the information user community.

Although the ERIC search on collective bargaining mentioned as an example seems to meet those expectations completely, it worked out so well only because of specific conditions that existed. All the tricks of computer technology would have been to no avail if there had not existed a data file in which there was a high probability of the information existing, if the information professional did not know of the existence of the file and how best to access it, if she had not had a purchase agreement with Lockheed and an appropriate terminal at her command.

If these facts seem elementary, it will surprise the computer community to report that great numbers of people with some knowledge of computing call our office without either knowing or being willing to learn these basic facts. And these callers are not limited to those whose knowledge of computing is purely peripheral. A gamut of callers assume that because they have access to the University's central computer facility they therefore have access to either ERIC or the other files on the Lockheed, SDC, Battelle, Medline, N.Y. Times, or other systems. It is an unpleasant job to convince them that their research allocation of $400 will not contribute one dollar to the cost of using any of the bibliographic data bases that are available in the interactive, on line mode that they are starting to hear so much about.

This type of problem occurs most often with seekers of information who are neither library nor computer professionals, although even with those groups some confusion exists. The real problem with the computer professionals, however, is that although they understand the concept of files and differing conditions through which they may be accessed, they often have little understanding of how bibliographic files are created; this makes it difficult to carry on a meaningful discussion on how to use them for information retrieval, whether on-line or in batch. Too many graduate computer scientists confuse the experimental work being done in automatic content analysis with the real world in which the analysis is done by human beings, and where data bases are created using descriptors, or subject terms, from a controlled vocabulary usually called a thesaurus.

And everybody is confused by the constant change in what is possible. Each data base has its own characteristics, which influence the way it can be accessed either at all or most effectively—however these characteristics change so rapidly that a file which one day can only be accessed by descriptor may, shortly after, be searchable for any word in the abstract (COMPENDEX is the most recent example). Also, new data bases are being added so frequently that a query which a few weeks ago might have been relevant to only one data base should now be considered from the point of view of, say, Dissertation Abstracts.

It is no more reasonable for me to expect computer scientists to keep up with every change in information retrieval than it would be for you to expect me to be completely *au courant* in the hardware. What I do ask in this area is that the computer professional be sufficiently educated about the conditions of on-line interactive retrieval so that he can play a positive role in the further development of this important sector of the computer economy. The exact nature of the role may vary all the way from not being a hindrance to affirmatively helping solve some of the problems in computing that affect many usages besides I.R.

The "not a hindrance" part of this request has two categories, the first of which is not to play pro in I.R. if your computer expertise is in other areas and, two, not to downgrade I.R. by giving it low priorities in terminal availability, budget for external computing, and training opportunities. Information professionals involved in on-line retrieval deserve terminals that are at least the equal of those assigned to the programmers in the organization. They have enough problems dealing with the intellectual problem of I.R. without in addition having to fight the equipment.

Both the "no hindrance" and the much more active involvement in on-line I.R. to be mentioned shortly are really problems that transcend a particular computer installation, and therefore belong in the domain of the entire computer economy as represented in the Conference. Any one manager has his/her own budget to optimize, his/her organizational hierarchy which determines priorities apart from any personal interest in I.R., and assorted additional internal computing problems that take immediate precedence over an operation that is on-line to an external system. The economic contributions of I.R. in any mode, and especially in the on-line interactive mode, are to the industry as a whole, and that is the reason for their being presented in a conference that represents the industry as a whole.

It would be easier to make the point if it were possible to make some claims for I.R. in terms of its proportionate value to the industry. What proportion of CPU time in the country is devoted to I.R.? What proportion of equipment costs, mainframe and peripherals? What is the proportionate use of programmers systems, analysts, data communications experts? Unfortunately, these questions are impossible to answer, even if put in absolute rather than proportionate figures.

Trying to arrive at some other measures for evaluating the role of on-line interactive information retrieval in the computer economy, some interesting estimates of absolute figures did emerge. The Lockheed Information System has eight billion characters of information on-line 14 hours a day (they start before dawn New

York time in order to accommodate European users). SDC contributes a like number of characters of information, and the bio-medical information systems (the National Library of Medicine and the SUNY Bio-Medical Network) a little over another eight billion together. These four systems are, therefore, making between 24 and 25 billion characters of information available currently in the interactive mode, most of them being used in the U.S. in locations remote to the residency of the data bases. Martha Williams of ASIDIC estimated recently that one million searches a year were being done on those four systems.

Since the figures as to sizes of the on-line systems were estimates of current availability obtained by informal telephone calls to appropriate people rather than from official reports, the same informality (combined with other conversations) persuades this writer that by the time this paper is actually presented, closer to 50 billion characters will be available for on-line interactive searching in the U.S.—especially if one takes into account the information systems available through Battelle, The Information Bank (N.Y. Times), the various ones now just coming up, and the normal growth in the size of files already in existence, as well as the expansion in amount of information in those files that can be searched interactively. There are also many international systems (the International Labour Organization, the World Health Organization, the U.N. Environment Centre, etc.) that seem on the horizon for direct access by U.S. information professionals, that will either contribute to the fifty billion figure or enlarge it.

The point to using characters of information as a measure of size is first its availability, and second its meaning to the computing community in what this means to the economy in terms of systems and programming support, hardware investment, and CPU utilization.

This approach does not solve the problem of how large information retrieval is proportionately to all other uses of the computer, including those for library administration, but it does indicate a contribution sufficiently large to make the point that the needs of information retrieval should be seriously considered in future developments of the industry. The size of information retrieval should also spur all information professionals, whether library trained or with other backgrounds, into demanding more than mere "no hindrance" policies mentioned earlier.

The most obvious need is for more education, education in which the computer professionals and the information professionals cooperate. Even with the development of such courses as those that have been offered in the City University's Professional Development Program for Library-Information Science, too few people have faced the problem of really bridging the gap between computer science and its bibliographic information systems implementation. Courses in programming only begin to bridge the gap; they must be supplemented by courses in how to measure the effectiveness of various kinds of programming. In the on-line remote environment, data communications can scarcely be ignored or confined to sociological discussion. The examples can be multiplied; the point is that educational efforts must be made by all segments represented by AFIPS because they are all involved in information retrieval.

A second "must" lies in the area of data storage. The closets are becoming filled, and will be more so when there are significant breakthroughs in input methods. The possible argument that information personnel already has at its command the very large amounts of bibliographic data already mentioned is not valid, if one starts to point out the large gaps in the bibliographic coverage of the literature, particularly in the social sciences. The systems need large capacity, and the librarians will have to learn to work with those larger capacities.

A third "must" lies in the area of referral vs. bibliographic services. There is too large a duplication of the latter—the overlap in data bases between SDC and Lockheed, for example, may be anti-monopolistic, but it uses capacity that might better service society's total communication needs by the provision of more referrals from file to file. As the number of files on-line grows, so will the necessity to refer even the expert to the correct file, and to instruct him/her properly in how to convert a question from one system to another (assuming that total file compatibility is an ideal rather than a complete reality).

A fourth "must", at least to this writer, is for a more imaginative use of terminals in information retrieval. We now have two possible ways of using the terminal on-line: letting the information roll off the screen or, if available, transferring it to a printing device. Some exploration might well be made of the divided screen, so that information (perhaps on strategy) can be kept in view while the materials to be scanned are rolling off the screen. The technology is presently used in another type of information system, and its application to I.R. use merits some study. Perhaps several screens, with or without divisions, will be needed to make full use of large volumes of material being on-line at once.

The topic of this session: "Enhancing Library Services Through Computer Technology" could have been interpreted as supplying the specifics for on-line interactive searching. However, those details are findable to the kinds of librarians and information center professionals who are likely to attend this meeting. This occasion should be used instead to broaden the strokes, to enlarge the possibilities, to engage the cooperation of the entire industry. It is my earnest hope that this has happened.

SCIENCE AND TECHNOLOGY

Computer and Data Base Architecture
Software
Computer Science
Applications of Computer Science

Developing application oriented computer architectures on general purpose microprogrammable machines

by TOMLINSON GENE RAUSCHER
NCR
Cambridge, Ohio

and

ASHOK KUMAR AGRAWALA
University of Maryland
College Park, Maryland

ABSTRACT

Surveying contemporary commercially available computers reveals a general incongruity between computer architectures and the problems the computers are being used to solve. Surveying the commercial applications of microprogramming reveals that microprogramming remains largely an alternative technique for manufacturer implementation of basic machine language instruction sets. With the large number of contemporary general purpose microprogrammable computers (especially minicomputers), the advantages of microprogramming are available to ordinary users for solving specific problems. From a pragmatic view, the architecture of a computer is defined by the microprograms resident in its control store. Changing the microprograms in a computer's control store therefore redefines its architecture. Architectures may be defined for specific problems by changing the microprogram in control store for each problem. As problems are represented by higher level language programs, compilers can *automatically* generate a microprogram for each higher level language program. The generated microprogram, when loaded into control store prior to program execution, defines an architecture that efficiently supports program characteristics. The advantage of this scheme is that it utilizes the power of microprogramming for each user's specific problem without forcing the user to comprehend the implementation complexities of a particular microprogrammable machine. This paper investigates several techniques for architecture redefinition via microprogramming.

INTRODUCTION

Motivation

Problem solving with the assistance of a modern general purpose digital computer generally involves a sequence of processes.

(1) The programmer writes a program to effect his algorithm in some higher level programming language.
(2) The analysis phase of a compiler for the programming language transforms the program into some intermediate language representation.
(3) The synthesis phase of the compiler transforms the intermediate representation into a machine language program.
(4) A microprogrammed interpreter (i.e., an emulator) directs the interpretation of the machine language program by the machine hardware to produce results.

This problem solving process may be considered to be a series of transformations on various representations of an algorithm to solve a particular problem. Most of the representations have been studied in some detail, and as a result there are well-known algorithm representations, higher level languages, and intermediate language representations. Similarly, most of the transformation processes have been studied in some detail, and as a result there are well-known programming techniques, formal language analysis techniques, semantic analysis and code generation techniques, and hardware transformation units. In contrast, the design of machine language instruction sets and the microinstructions that interpret them have been given relatively little consideration.

> Some things have not altered, or only slightly, so, like the Model T, they have remained invariant, upright, slow, inefficient and immune to the winds of progressive technological improvement. . . . I am referring to the basic form and rigid format of our instruction sets.[1]

The instruction sets of many contemporary computers were designed with hardware realization as the foremost constraint. Little attention has been given to the types of operations the computers will perform. As a result, machine language representation of program constructs are often awkward, and the efficiency of machine code for programs is seldom high. "The majority of the . . . required work is involved with placating the computer, while a relatively small portion of the work is actually applied to the job."[1] "In this regard it is interesting to note that . . . most of the instructions executed, even in scientific applications, are used for "housekeeping' details."[2]

Microprogramming was formulated more than twenty years ago as a systematic alternative to the usual somewhat *ad hoc* procedure for designing the control section of a digital computer. Although microprogramming has been used commercially for more than ten years, it generally remains a technique for the implementation of basic machine language instruction sets. Recent developments, such as the availability of fast writable control stores, have sparked investigation into other applications, however, these are mainly special purpose and not immediately generalizable.[3]

The advantages of microprogramming have been mainly engineering aspects: low cost, flexibility, and ease of development and maintenance. Because microprograms interpret instruction sets, microprogramming can be applied to general purpose problem solving to overcome inefficient machine languages.

Environment

Problems to be solved will be represented in procedure oriented higher level programming languages. For the higher level language there is a compiler consisting of two parts:

(1) an analysis phase which transforms the higher level language program into an intermediate language representation and,
(2) a synthesis phase which transforms the intermediate language representation into machine language.

Since a majority of optimization techniques are machine independent, the first compilation phase performs all the machine independent optimizations (such as folding, eliminating redundant operations, moving invariant operations, reducing strength of operations, and eliminating dead variables and assignment operations) before it produces the final intermediate representation of the higher level language program. A general purpose dynamically microprogrammable computer interprets the compiler generated machine code to produce results. "Dynamically microprogrammable" means that the computer has a fast writable control store (much faster than main memory) that can be loaded under program control. Once compiled and debugged, programs will be executed repeatedly. Figure 1 summarizes the problem environment.

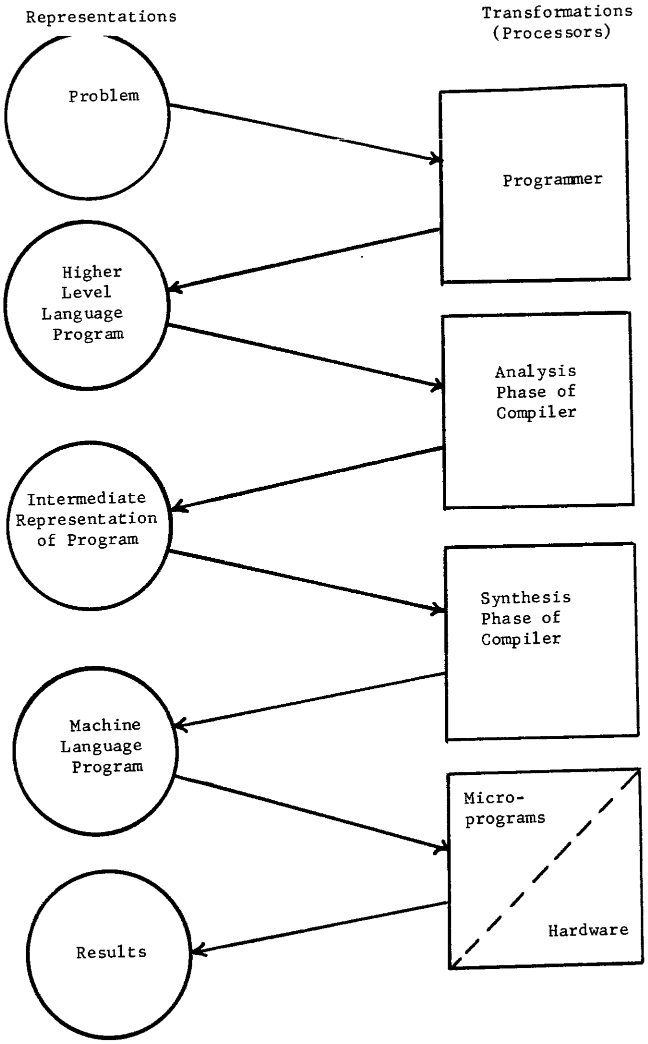

Representations
Transformations (Processors)
Problem
Programmer
Higher Level Language Program
Analysis Phase of Compiler
Intermediate Representation of Program
Synthesis Phase of Compiler
Machine Language Program
Micro-programs
Hardware
Results

Figure 1—Problem solving with the assistance of a modern general purpose computer

Problem definition

Basically, the problem is that computers understand, i.e., perform, machine oriented instructions, and as a result problems expressed in higher level languages must be transformed into machine language programs. The resulting machine language representations of programs are almost always inefficient, and hence the process of computer assisted problem solving in the environment discussed is inefficient. The genesis of this problem has been observed in the constraint of contemporary computer instruction sets to machine oriented operations rather than problem oriented operations:

> When engineers begin to seek more efficient encodings for commonly used sequences of instructions, progress toward the modern computer may begin.[4]

It has been observed that contemporary machines use inefficient machine languages to represent algorithms. More generally, the architectures of computers do not efficiently support programs being run. By computer architecture we mean the attributes of a computer as seen by the programmer, i.e., the conceptual structure and the functional behavior. The conceptual structure comprises such considerations as the memory hierarchy (registers, main memory, external store, etc. and their logical connections), the representation of data and instruction sets, and addressing schemes. The functional behavior is determined by the machine language instruction set, the instruction interpretation process, and the instruction execution process. The architecture of a computer is to be distinguished from its implementation which includes the entire set of machine registers (including the various memories), the physical connections among the machine registers, the mapping of the conceptual structure onto the physical structure, and the implementation of machine language instructions. The efficiency of solving a particular problem on a computer depends primarily on the degree to which the architecture of the computer supports the problem primitives. The primitives of major concern in expressing a problem are the data structures inherent in the problem, the operations (transformations) that manipulate the data structures, and the flow of control or sequencing among the operations. These primitives are employed in the formation of a program which represents an algorithm to solve the problem. Since there is a wide range of problems and algorithms (programs) for solving them, each problem would be handled most efficiently by a computer whose architecture supports the primitives of the algorithm that solves the problem. To realize a variety of algorithms efficiently, it is therefore necessary to define architectures for the different problems.

With a given control store size, the factors of concern that measure the performance of a machine in executing a program are the time required to execute the program and the main memory space required to store the program. While time and space are also functions of the machine realization (e.g., the technology from which the memories are fabricated), here we are interested only in the architectural changes that affect the time and space required to execute a program.

The problem under study may be summarized as follows:

Given a higher level language program and a dynamically microprogrammable computer with a fixed size control store, design an architecture (including the instruction set into which the program can be translated) that (when interpreted by the machine) minimizes the execution time and main memory requirements for the program.

Related work

A design approach that is presently being popularized commercially is language directed computer architecture. As early as 1967, McKeeman noted the absurdity of expecting modern digital computers (which are designed to be very fast automatic desk calculators) to be suitable hosts for supporting such diverse applications as operating systems and compilers.[4] In the Burroughs B1700 computer,[5] there is a novel "S-Language" for each higher level language. The S-Languages are interpreted by microprograms that reside in main memory or in a faster writable control store. While language directed computer architecture shows considerable improvement over the traditional single machine language architecture, there is still considerable disparity between language directed instruction sets and ideal instruction sets for programs.

Microprogramming has been used to improve the performance of programs in several application areas.[3] Inserting microprograms into control store to support a class of problems results in application or environment oriented architectures. The success of this approach depends on the size of the application area. Good results may be obtained for small, well defined application areas.

The approach developed subsequently is an extension of the language directed and environment directed approaches that does not rely on characteristics of particular languages or characteristics of particular environments.

APPROACH

Implementing problem oriented architectures via microprogramming

Within the given environment there are alternative approaches to implementing performance improvement by redefining computer architecture for individual problems. Of concern are the flexibility of the implementation and the binding time of the redefinition.

Since it is necessary to define an architecture for each problem, one approach to implementing architecture definition is to design for each problem a special purpose machine based on the structure of the algorithm and its intrinsic data structures. This approach has been used by Shay[6] in designing machines whose algorithms exhibit natural parallelism. Such an implementation, however, yields a special purpose machine for each class of similarly structured problems. The expense in building a special purpose machine can generally be justified only in dedicated situations, such as real time radar signal processing.

From a pragmatic view, the architecture of a general purpose computer (especially the machine language instruction set) is defined by microprograms resident in the control store and the associated implementation which the microprograms control. The microprograms interpret the machine language instruction set and thus define the instruction format, the instruction set, the instruction interpretation process, and the addressing schemes. Since microprograms control the referencing of machine registers, the microprograms also determine the conceptual structure of a computer's architecture. Changing the microprograms in the control store of a machine therefore redefines the architecture of a computer. Since the problem environment under consideration specified dynamically microprogrammable computers (to reflect capabilities of a large number of contemporary machines), changing the microprogram is a simple task.

Dynamic redefinition

Defining architectures for specific problems requires changing the microprogram in control store for each problem. As problems are represented by higher level language programs that are analyzed by compilers, it is natural to have the compiler generate microprograms.

The approach then is to replace the code generation phase of a compiler by a procedure that performs the following operations:

(1) generates a microprogram that defines an architecture for efficiently supporting the higher level language program being compiled (this includes interpreting the machine language programs of the architecture) and
(2) generates the machine language program for the defined architecture that represents the higher level language program being compiled.

To execute the compiled program, the generated microprogram and machine language program are dynamically loaded into control store and main memory to redefine the architecture of the general purpose host machine. This approach is called dynamic problem oriented redefinition of computer architecture via microprogramming.

Because control store is such a limited resource, translating each higher level language program directly into microcode is not feasible; there would be too many microinstructions for the control store to hold. The compiler, therefore, must still generate a "machine language" program that, when interpreted, effects the algorithm. This means that there is an almost infinite variety of "machine languages".

Effect of control store availability

Since control store is a limited resource, the performance of the procedure that generates microprograms depends on the amount of available control store. Usually control store is small compared to main memory, so the number of generated microinstructions cannot be large. This means that the number of special instructions is small and that most program operations are represented by instructions in a basic machine language.

The instruction sets of the large majority of contemporary computers are redundant in that many operations can be implemented by several different sequences of machine language instructions. As an early example of redundancy in instruction sets, one research group found that "from the programming point of view, little flexibility would be lost if the set of instructions on the CDC-3600 were reduced to less than ½ or ¼ of the instruction options now available."[2] Similar remarks apply to almost all contemporary computers.

Since machine language instruction sets are so redundant, an obvious way to obtain more control store for use in architecture redefinition is to remove the microprograms that interpret superfluous machine instructions. A machine instruction is superfluous if none of the operations in the program need be implemented using that instruction. While this method can provide more control store, it has disadvantages. Upon removal of some machine language instructions, the instruction set, though still complete, may be very inefficient. That is, it may require several machine language instructions to implement an operation that formerly required just a few machine language instructions. The instruction set may not be well rounded, i.e., some operations could be easily implemented while others would be implemented inefficiently.

A better approach would be to design a "kernel" machine language which is simple, complete, balanced, and does not have trivial redundancy. Operations in the kernel machine language should include operations for flow of control constructs, arithmetic and logical operations to manipulate the primitive data types, I/O operations, etc. The actual language of course depends on the environment in which programs will run, e.g., for simple business applications there is little need for data types like real and complex or for data structures like sets and lists. Even within a single computer system it is feasible to have a variety of kernel languages for application to a variety of problem types. The microprogrammed implementation of kernel machine language instruction sets uses only a small portion of control store. It is not minimal but reasonable in its implementation of program operations.

METHODS FOR DEFINING NEW INSTRUCTION SETS

Instruction sequence method

Many researchers have observed that instructions in the static representation of a program fall into natural sequences.[2,7] That is, having seen an instruction at a certain location in the program, some instructions are more likely to follow it than other instructions. Contemporary machine languages generally do not take advantage of this dependence. When considering particular programs to be run on a computer, the observation about dependence between pairs of machine language instructions can be extended to operations in intermediate language programs. It is characteristic of programs to use the same sequence of operations at many points. This observation is the basis for the sequence method of defining new "machine language" instructions.

The sequence method analyzes the intermediate language representation of a program to find all sequences of instructions and the number of times each sequence appears. The method then compares the microprogrammed implementation of each sequence to its machine language implementation. The microprogrammed implementation of a sequence will show improvement over the machine language implementation because it represents a sequence by a single operation code instead of many operation codes. The sequences that yield the most savings are selected as new "machine language" instructions.

As a result of developing new instructions that replace several instructions, the sequence method will also reduce program execution time. The improvement does not usually approach optimality because the method does not take into account the interaction of sequences, especially the dependence between sequences and their subsequences. As the sequence length increases, the number of occurrences of different sequences decreases. Thus there is a tradeoff in implementing sequences of different lengths as new instructions. While representing each occurrence of a long sequence by a new instruction reduces memory size (and execution time) more than replacing its constituent short sequences does, it may be advantageous to use the available control store to interpret instructions that represent short sequences because they appear more frequently in the intermediate language program.

Program structure method

The intermediate language representation of a higher level language program has been used as input to the architecture redefinition process. This static description of the program does not directly provide information about the run time behavior of the program. To minimize execution time, however, the architecture redefinition process requires knowledge of the run time behavior.

One way of learning about the run time behavior of a program is to execute it many times with several representative data sets. By monitoring these executions, information about the run time behavior may be gathered for analysis. An alternative method is to estimate the dynamic behavior using knowledge of input data and statistical techniques. The program structure method for defining new instructions assumes knowledge about the execution behavior of a program. More specifically it assumes knowledge of the frequency of execution of program blocks, i.e., sequences of operations that are executed sequentially with no branching or selection. (Thus if the first operation in a block is executed, all the operations will be executed.)

For each block, the program structure method compares the execution times of the microprogrammed implementation and the machine language implementation. The microprogrammed implementation will require less execution time than the machine language implementation because few instructions need to be fetched and decoded. The method then multiplies the difference by the number of times the block will be executed.

The blocks that yield the most total execution time improvement are selected as new "machine language" instructions.

Some empirical evidence has been reported to support the efficacy of this method. In one study, Knuth "found that less than four percent of a program generally accounts for more than half of its running time."[8] Given that forty percent of the time involved in performing machine language instructions is devoted to fetching and decoding the instructions, Knuth's findings imply that this architecture redefinition method can easily reduce execution time by twenty percent.

A combined method for architecture redefinition

The two methods presented for defining applications oriented architectures may be criticized for using only some of the available information. The sequence approach defines instructions that appear globally throughout a program, however, it does not consider program behavior at execution time. The structure approach defines operations that are local to a part of the program, however, it does not consider the operations that constitute the blocks throughout the program. For these reasons we consider a combined approach.

As in the sequence method, the combined method first finds the different instruction sequences, the places they appear in the program, and the differences between the execution times of the microprogrammed implementation and the machine language implementation. For each sequence, the combined method then multiplies this savings by the expected frequency of execution of the sequence. For all occurrences of a sequence throughout a program, these products (savings times execution frequency) are added. The resulting sums represent for each sequence the total execution time saving when the program is run. Those sequences with the highest savings can then be microprogrammed as new "machine language" instructions.

Defining new architectures in this way reduces program execution time for two reasons. First, fewer "machine language" instructions need to be fetched from main memory and decoded because several operations have been represented by a single new operation code. Second, the microprograms that interpret the new instructions may be optimized. The amount of space required to store the program will also be reduced because several sequences of machine language instructions are replaced by single new instructions.

EXAMPLE IMPLEMENTATION

Environment

An example implementation has been developed to demonstrate the practicability of problem oriented redefinition of computer architecture by microprogramming. The implementation consists of:

(1) the analysis phase of a compiler for a higher level structured programming language,
(2) a procedure that takes the output of the analysis phase and generates microprograms that define an architecture to support the higher level language program being compiled and also generates machine language instructions to represent the higher level language program, and
(3) a simulator that interprets the microinstructions of the host machine.

The higher level language chosen for the implementation was ULP,[9] a language that is simple in structure and general in scope, yet maintains the philosophy of structured programming. The simulator was developed for the Digital Equipment Corporation PDP-11/05 (See Figure 2) with some minor differences. The most important difference was the assumption of a writable rather than a read only control store. This machine was chosen because good documentation was available on the microinstruction format and the machine language emulator, the microprogram level architecture was fairly simple and general (if somewhat inefficient for general purpose use), and the machine size epitomizes situations where architecture redefinition would be beneficial. The simulator simulates microinstructions that interpret machine language instructions. The architecture redefinition program takes the intermediate language representation of a higher level language program generated by the ULP compiler and performs the combined procedure using a kernel language as described previously.

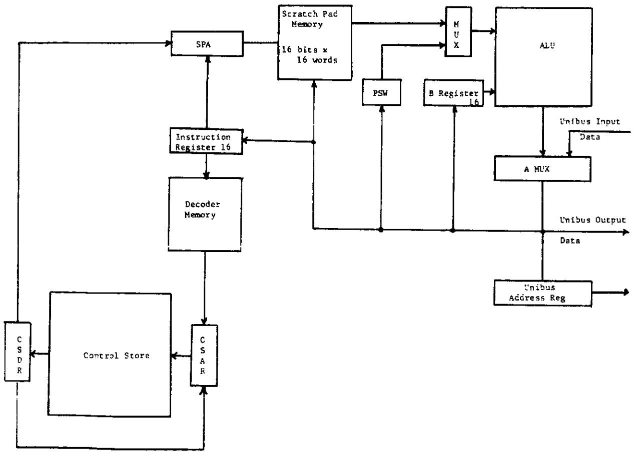


Figure 2—Organization of the PDP-11/05

With the ULP language and compiler, the architecture redefinition program, and the PDP-11/05 simulator it is easy to compare the performance of a program as executed on the PDP-11 architecture to the performance of the program as executed on the redefined architecture.

Simulation results

For the simulations, two ULP programs were selected. The first program is a part of a programming system developed to design telephone exchanges, and the second program computes operator precedence relations for an input grammar. Each program was simulated nine times on each of two different input data sets, for a total of thirty-six simulation runs. Of the nine simulations for a program on one set of data, the first represented the standard PDP-11 architecture interpreted by the PDP-11/05 emulator. The remaining eight simulations represented the kernel and redefined architectures (generated by the architecture redefinition program) using control store sizes ranging from 64 to 512 words in increments of 64 words. Each simulation recorded the execution time, the number of machine language instructions executed, and the number of microinstructions executed.

As illustrated in Figure 3, execution time for the simulated programs decreased as the amount of available control store increased. Especially important are the decreases in execution time of the redefined architecture using 256 control store words compared to the PDP-11 architecture (which also uses 256 words of control store). The execution time improvement of more than twenty-five percent may be somewhat modest because small data sets were used to keep simulation times reasonable (less than ten minutes on a UNIVAC 1108). In retrospect, larger (and more typical) data sets would have resulted in the programs spending more of their time in frequently executed program loops. As these loops contained new instructions, significant additional time savings would accrue.

Figure 4 illustrates the effect of the new instruction set on program execution. It shows that the number of executed "machine language" instructions on the new architecture was less than half of the number of machine language instructions executed by the PDP-11.

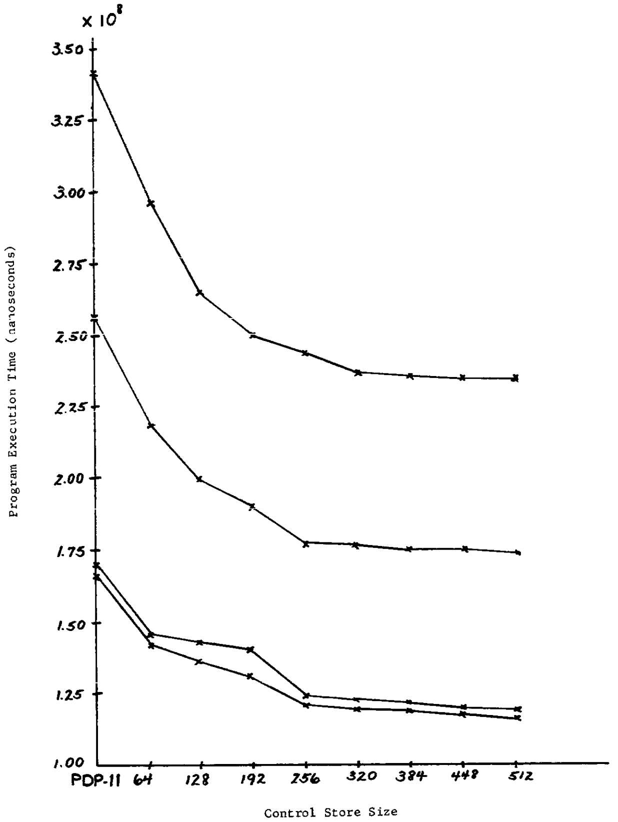


Figure 3—Program execution time vs. control store size for simulated programs

The defined architectures were indeed oriented toward the particular applications.

SUMMARY

The purpose of this paper was to investigate techniques involved in defining architectures to support different problems solved on general purpose computers. In the selected approach microprograms are generated by a compiler and loaded into the control store of a computer thus redefining the machine architecture for the particular problem program.

The first technique for designing new instruction sets was motivated by the observation that instruction executions exhibit repeated sequences of operations. The sequence approach searches through the intermediate language representation of a program and represents commonly occurring sequences as new machine instructions. The second technique was motivated by the observation that different program parts are executed with different frequencies. From the dynamic behavior, the program structure approach represents the most frequently occurring program blocks as new instructions. The architecture redefinition procedure developed to solve the problem combined the techniques used in the sequence and program structure methods.

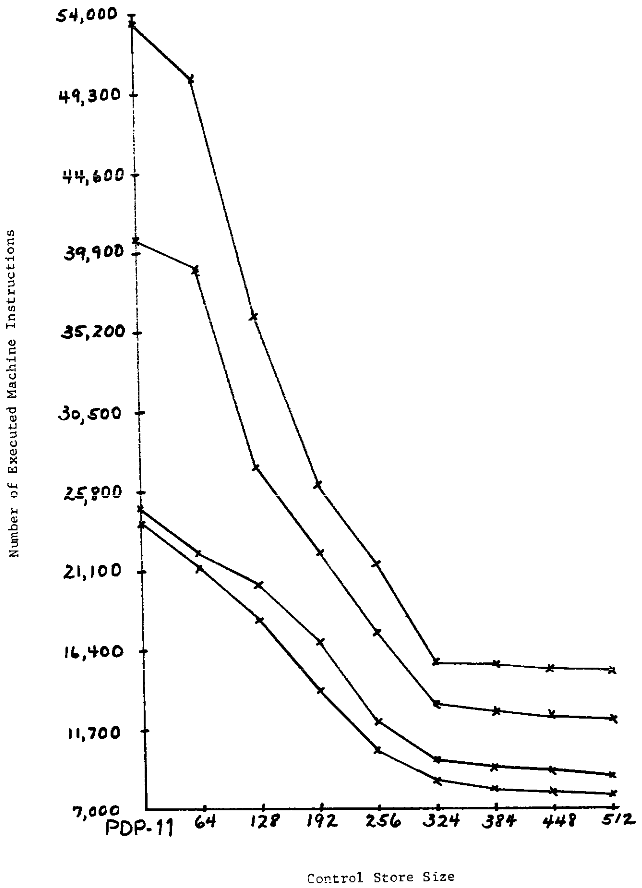


Figure 4—Number of executed machine instructions vs. control store size for simulated programs

An example implementation using this procedure was developed. Simulations provided empirical verification of the procedure in improving performance by defining architecture oriented toward particular applications.

REFERENCES

1. Church, Charles C., "Computer Instruction Repertoire—Time for a Change," *1970 Spring Joint Computer Conference Proceedings*, AFIPS Press. Montvale, New Jersey, pp. 343-349.
2. Foster, Caxton C., Robert H. Gonter and Edward M. Riseman, "Measures of Op-Code Utilization," *IEEE Transactions on Computers*, Volume C-20, Number 5, May 1971, pp. 582-584.
3. Agrawala, Ashok K. and Tomlinson G. Rauscher, *Foundations of Microprogramming: Architecture, Software, and Applications*, ACM Monograph Series, Academic Press, New York, 1976.
4. McKeeman, W. M., "Language Directed Computer Design," *1967 Fall Joint Computer Conference Proceedings*, AFIPS Press, Montvale, New Jersey, pp. 413-417.
5. Wilner, W. T., "Design of the Burroughs B1700," *1972 Fall Joint Computer Conference Proceedings*, AFIPS Press, Montvale, New Jersey, pp. 489-497.

6. Shay, Barry P., *A Microprogrammed Implementation of Parallel Program Schemata*, Ph.D. Dissertation, University of Maryland, Department of Electrical Engineering, 1975.
7. Saal, Harry J. and Leonard J. Shustek, "Microprogrammed Implementation of Computer Measurement Techniques," *Fifth Annual Workshop on Microprogramming Preprints*, ACM, September 1972, pp. 42-50.
8. Knuth, Donald E., "An Empirical Study of FORTRAN Programs," *Software—Practice and Experience*, Volume 1, 1971, pp. 105-133.
9. Mills, David L., *Structured Programming and Compiling in a Minicomputer Environment*, Computer Science Technical Report Series, Technical Report TR-339, University of Maryland, October 1974.

A pipeline polish string computer*

by GERARD G. BAILLE and JEAN P. SCHOELLKOPF
Computer Architecture Group
Grenoble University, France

ABSTRACT

This paper describes a new computer organization which allows the pipeline execution of a polish-string code. The central characteristic of the proposed organization is a FI-FO queue, that holds values just accessed by an Access Station, until they are used as operands by an Execution Station for an arithmetic or logical operation. Operands are taken out of the queue by means of two pointers whose modifications are managed by a Control Station. This new computer organization is capable of high performance, since access to variables and execution of operations are performed in parallel with control functions required by the input program. A solution to access conflicts is proposed, using a content addressable memory that holds the names of the variables whose modification is deferred. This architecture is currently in application for the design of a high-level PASCAL computer.

INTRODUCTION

Design of high performance computers can be achieved using the technique that is called "pipeline" design, characterized by the fact that concurrent operations are supported by the machine.[10] A high-level instruction can be initiated within a module, and in the same time the other modules are executing some operation related to a preceding instruction. The IBM 360/91[1] and the CDC 6600[2] are two examples of pipeline execution. Efficiency of pipeline computers depends strongly on the way that they are programmed. Many attempts were made to solve this problem and several techniques are proposed for generating optimized code in terms of pipeline execution.[3-5] These techniques imply an important amount of preprocessing, which does not always justify the complexity of pipeline execution.

The solution proposed in this paper is based on a *natural* decomposition of the work to be executed in a pipeline manner. The first aspect of a natural decomposition is related to the kind of language proposed to the machine. Tomasulo[9] gives an efficient algorithm for exploiting a pipeline architecture, but its application is limited to a low level language. On the other hand polish-string code appears as best suited for the execution of high level language.[6]

Stones[7] proposes a pipeline architecture for a push-down stack computer, but he suggests to translate the input polish-string code into three address instruction code. This paper shows, in a first section, how polish-string code can be directly executed by a pipeline computer. The second aspect of a natural decomposition, as shown by Abrams,[8] is related to the decomposition of input string code execution into three *natural* processes that are control, access to operands and execution of operators. It would be interesting to have the three above processes concurrently running in a pipeline manner: the execution of an instruction would be initiated within the control station, and in the same time preceding instructions would be currently in process either within the access station or within the execution station. Such a pipeline organization is made possible using a FI-FO queue instead of a push-down stack as a work area for expression evaluation. Operands are accessed from the queue by means of two pointers whose modifications are controlled by the control station which generates extra-orders. The generation algorithm is presented and the rate of extra-orders is evaluated in the first section.

The second section of this paper gives the general architecture of the pipeline polish string computer. Concurrency between access to operands and evaluation of expression to be assigned to variable leads to the well-known problem of reading the value of a variable whose modification is not yet performed. This problem, that is called "dependency" problem, is solved at execution time using a content-addressable memory which holds the name (not the address as in classical pipeline computers) of the variables whose modification is deferred.

A brief summary is given at the end of this paper, which shows the possible applications of the proposed pipeline architecture.

* The research reported on this project is sponsored by the French Comity for Computer Science Research under contract SESORI No. 74-156.

PIPELINE EXECUTION OF POLISH STRING CODE

This section first explains the choice of a FI-FO queue as a work area for evaluation of polish string. A model is introduced for polish string expressions which allows the definition of two pointers for operand access. The operator set is reduced to monadic and diadic operators. The management of the two pointers is controlled by special extra-orders generated by a Control station. The generation algorithm is presented and discussed.

The central characteristics of the pipeline execution is that the input polish stream is analyzed by a processor which generates two parallel instruction streams towards two concurrent processors specialized the one for access to operands, the other for execution of operations.

Polish string evaluation

A polish string can be seen as a sequence of groups, each group being a sequence of operands followed by a sequence of operators. An operand is either an immediate value or the internal name of a variable (for example, lexical level and offset) which allows the calculation of the variable address in main memory.

example:

$$\underbrace{[A,B,C,D,+,*,]}_{\text{group 1}}\quad\underbrace{[E,F,+,]}_{\text{group 2}}\quad\underbrace{[G,*,+,-]}_{\text{group 3}}$$

The evaluation of such a polish string requires the use of a working storage classically organized as a push-down stack.

All the operands of a given group are pushed, one after the other, into the stack, in the same order as in the input string, next operators are sequentially executed, poping the two topmost elements from the stack, and pushing the result onto the stack.

When all the operators of the current group are executed, the operands of the next group are pushed onto the stack, and the above process is performed again.

Therefore, two processes appear when evaluating a polish string:

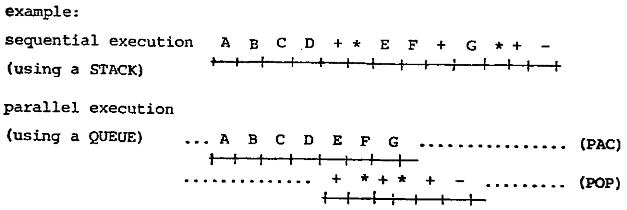
- —an ACCESS process which stores the operands into the working storage (either from the string when immediate access, or from main memory in the other case)
- —an EXECUTION process which executes the operators, taking its operands out of the working storage and storing intermediate results into it.

Using a push-down stack as working storage implies the sequentiality of the two above processes. The aim of this paper is to propose an evaluation method which allows parallel execution between ACCESS and EXECUTION process: parallelism is made possible using a FI-FO queue instead of a PUSH-DOWN stack.

Organization

The above two processes are executed by two independent processors. Let PAC be the processor which stores the operands into the FI-FO queue, and POP be the processor which executes the operators, taking its operands out of the queue.

Hence, when POP is executing an operator, PAC can ACcess to a new operand and store it into the queue.


example:
sequential execution
(using a STACK)
A B C D + * E F + G * + -
parallel execution
(using a QUEUE)
... A B C D E F G .... (PAC)
.... + * + * + - .... (POP)

Such an organization requires a third processor which is called PAN, whose work is the ANalysis of the input polish string in order to generate instructions for both processors (PAC and POP).

The architecture is given by Figure 1.

THE EVALUATION PROCESS USING A FIFO QUEUE

Variables whose names appear in the input polish string are stored into the FIFO queue by the Access Processor. The sequence of the access instructions is the same as the sequence of the variable names in the input string. So, we can define the relative location of any variable in the queue, depending on its relative location in the input string.

Let $\bar{S}=(V_1^1, \ldots, V_1^{n_1}, O_1^1 \ldots O_1^{m_1}, \ldots, V_p^1 \ldots V_p^{n_p},$

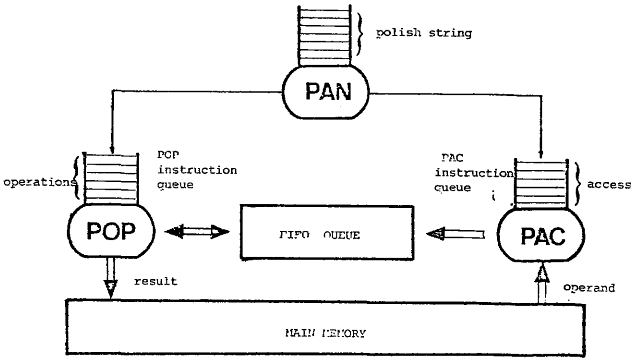


Figure 1

$O_p^1 \dots O_p^{m_p}$) be the input string, where V_i^j is a variable name, and O_i^j is an operator name.

If the relative location of the first variable V_1^1 is equal to 1, then the location of any variable V_i^j is given by the formula:

$$1(V_i^j) = \sum_{r=0}^{i} n_2 + j, \text{ with } n_0 = 0$$

How can we access to the operands located in the queue

The evaluation of polish string is a sequence of monadic or diadic operators execution. In a first approach, let us consider that all the operators are diadic ones. We must associate to each operator its two operands, whose locations are partly defined by the next two rules:

RULE 1: if O_i^1 is a diadic operator, then its second operand is variable $V_i^{n_i}$, and its first operand is either variable $V_i^{n_i-1}$ if $n_i>1$, or the result of the immediately preceding operator $O_{i-1}^{m_{i-1}}$ if $n_i=1$.

RULE 2: for $j=2$ to n_i, the second operand of operator O_i^j is the result of the preceding operator O_i^{j-1}.

We see that the variables are not referred to as operands in the same order as in the input string:

for each group $G_i = (V_i^1, \dots, V_i^{n_i}, O_i^1, \dots, O_i^{m_i})$ the first accessed variables are $V_i^{n_i}$ and $V_i^{n_i-1}$ as operands for operator O_i^1, the last accessed variable V_i^1.

Hence, it is idle to pull the operands out of the queue in a FIFO mode, since it should be necessary to store them again into another memory, organized as a Push-down stack.

So, we propose to use the queue as a working storage from which operands are accessed by means of two pointers, and into which intermediate results are stored during the evaluation process.

Definition of two pointers

Let us define P1 and P2 as two pointers which hold respectively the address of the first and second operand of any diadic operator during the evaluation process.

As an example, let $S = (V_1^1, V_1^2, V_1^3, O_1^1, O_1^2, V_2^1, V_2^2, O_2^1, O_2^2)$ be the input string. The variables are stored into the queue in the following manner:

	1	2	3	4	5	
...	V_1^1	V_1^2	V_1^3	V_2^1	V_2^2	....
	↑ first-in				↑ last-in	

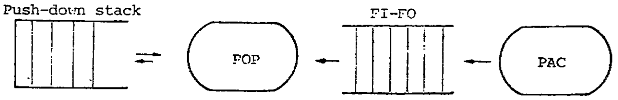


Figure 2—Classical organization

The input polish string defines the sequence of operations as follows:

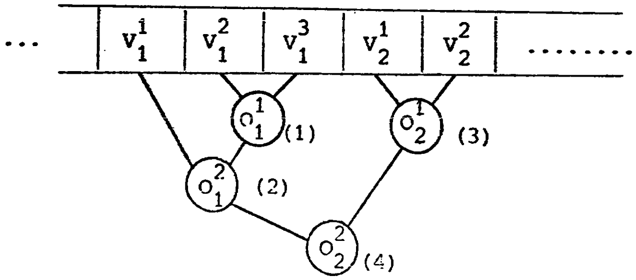


Figure 3

We must generate orders for initializing P2 on variable V_1^3, P1 on variable V_1^2 before executing operator O_1^1, whose result is immediately used as second operand for the next operator O_1^2, whereas pointer P1 must be decremented by one in order to give the address of variable V_1^1 as first operand for operator O_1^2.

The next diagram shows the successive states of the queue and the successive values of pointers P1 and P2.

Initialization of pointers

Both pointers P1 and P2 must be initialized after execution of the last operator of group G_{i-1} $(O_{i-1}^{m_{i-1}})$ and before execution of the first operator of the next group $G_i(O_i^1)$. We know that pointer P2 gives the address of the location which holds the results of operator $r(O_{i-1}^{m_{i-1}})$: this address is equal to $1(V_{i-1}^{n_{i-1}})$. We must assign to P2 the address of the second operand of operator O_i^1, which is variable $V_i^{m_i}$ (from rule 1), whose address is equal to

$$1(V_i^{n_i}) = \sum_{r=0}^{i} n_r = \sum_{r=0}^{i-1} n_r + n_i$$
$$= 1(V_i^{n_i-1}) + n_i$$

So we must increment the previous value of P2 by n_i. Moreover, pointer P1 must hold the address of the first operand of operator O_i^1, which is either variable $V_i^{n_i-1}$ if $n_i>1$, or the result of operator $O_{i-1}^{m_{i-1}}$ if $n_i=1$.

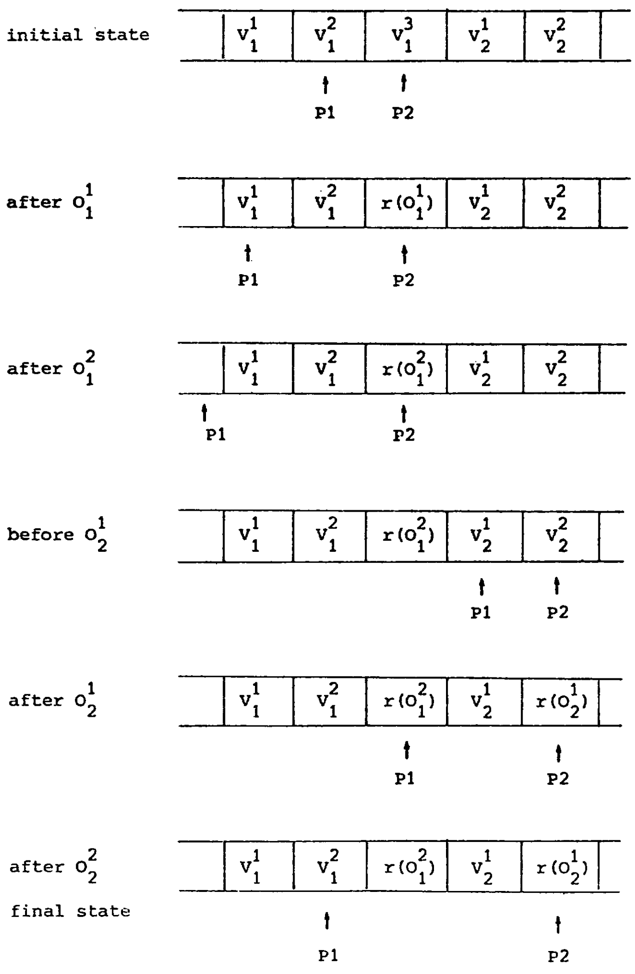


Figure 4

In both cases:

$P1 = l(V_{i-1}^{n_{i-1}}) = l(V_i^{n_i}) - 1 = P2 - 1$ if $n_i > 1$,

$P1 = l(V_{i-1}^{n_{i-1}}) = l(V_i^{n_i}) - n_i = P2 - 1$ if $n_i = 1$

So, we define an EXTRA-ORDER, generated between the execution of $O_{i-1}^{m_{i-1}}$ and the execution of O_i^1, called UP (n_i), which is interpreted as:

$$\boxed{P2 \leftarrow P2 + n_i \; ; \; P1 \leftarrow P2 - 1}$$

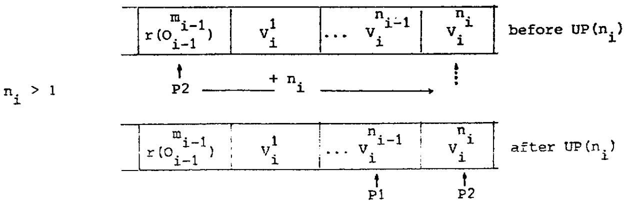
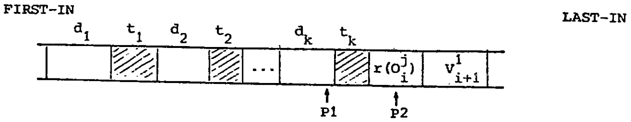


Figure 5

The intermediate states of the queue

Each time a diadic operator is executed, the location which holds the first operand will be no more accessed: we say that a "hole" is created or the the location becomes "empty". So, during evaluation process, the state of the queue is defined as a sequence of empty locations followed by full locations which hold either not yet accessed variables or intermediate results. We associate to the state of the queue, after execution of any operator O_i^j a finite sequence

$\{(d_1, t_1), (d_2, t_2), \dots\dots, (d_k, t_k)\}$, where each d_i is the length of a full location sequence, and each t_i is the length of an empty location sequence



The successive values of pointers P1 and P2 can be defined related to the state of the queue:

P2 must hold the address of the result $r(O_i^j)$, and P1 must point to the full location downstream of the location pointed by P2.

Each time an operator is executed, a new hole is created, therefore the last element of the sequence $\{(d_i, t_i)\}$ must be modified: t_k is incremented by 1 and d_k is decremented by 1. During execution, pointer P1 will be decremented by one. However, d_k may become zero, in which case pointer P1 must be decremented again by a value equal to t_{k-1}, in order to point to the first full location downstream of P2.

In the same time, the queue state is updated, by suppressing the last element

$t_k \leftarrow t_{k-1} + t_k \; ; \; k \leftarrow k-1$

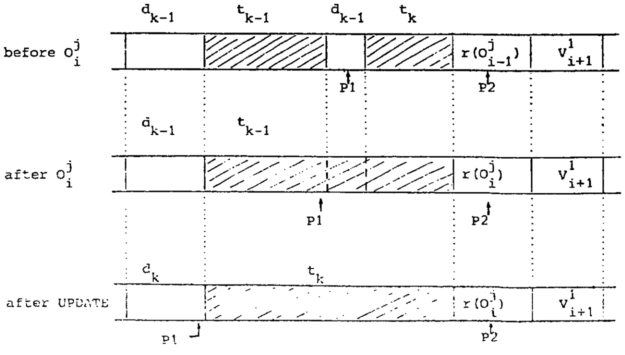


Figure 6

Conclusion—An extra-order called DOWN must be generated by the PAN processor, when the queue state is updated: its execution is defined by: $P1 \leftarrow P1-n$, where n is equal to t_{k-1}, when the order is generated.

Modification of the queue state before execution of a new group

Between execution of operator O_{i-1}^{m-1} (the last operator of group G_{i-1}) and execution of operator O_i^1 (the first operator of group G_i), an extra-order UP must be generated. This movement of pointers corresponds to a modification of the queue state: a new element (d_{k+1}, t_{k+1}) is created, initialized as follows: the new group G_i is defined as a sequence of n_i operands, hence $d_{k+1} \leftarrow n_i$, and zero hole has been created, hence $t_{k+1} \leftarrow O$.

Conclusion—An extra-order called UP must be generated by PAN processor in order to update the queue state before the first operator of a new group. Its execution is defined by:

$$P2 \leftarrow P2+n \ ; P1 \rightarrow P2-1$$

where n is equal to $d_k = n_i$, when the order is generated.

GENERATION OF EXTRA-ORDERS FOR THE MANAGEMENT OF THE POINTERS

The proposed evaluation process using a FIFO queue requires the generation of extra orders for the management of the pointers. As shown above, three kinds of orders must be sent to the execution processor:

—the first kind consists in all the monadic or diadic operators, whose execution is defined by:

$$Q(P2) \leftarrow Q(P1) <Op> Q(P2) \qquad \text{(diadic operator)}$$

or

$$Q(P2) \leftarrow <Op> Q(P2) \qquad \text{(monadic operator)}$$

The arithmetic or logical diadic operation is followed by a modification of pointer P1:

$$P1 \leftarrow P1-1$$

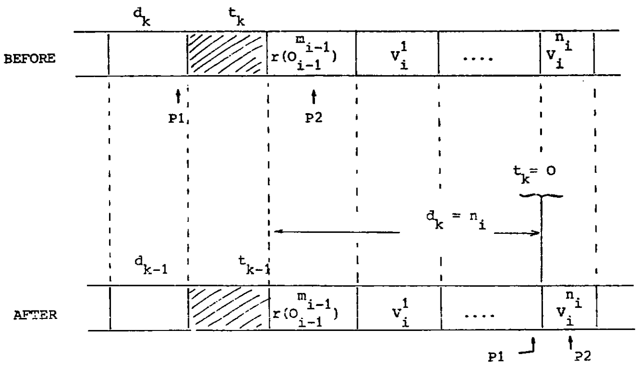


Figure 7

—the second kind is the UP(n) extra-order which is executed when a new group is entered. It is defined as:

$$P2 \leftarrow P2+n;$$
$$P1 \leftarrow P2-1.$$

—the last kind is the DOWN(n) extra-order, which consists in the updating of pointer P1; it is defined as:

$$P1 \leftarrow P1-n$$

Evaluation of the number of extra-orders to be generated

The number of extra-orders $(n_{up}+n_{down})$ to be generated only depends on the input string.

Let $S = (V_1^1, \ldots, V_1^{n_1}, O_1^1, \ldots, O_1^{m_1}, \ldots, V_p^1, \ldots V_p^{n_p}, O_p^1, \ldots, O_p^{m_p})$ be the input string.

Such a string contains

$N = \sum_{r-1}^{p} n_r$ variable names, hence N access instructions for the access processor PAC, and

$M = \sum_{r-1}^{p} m_r$ operators, hence M instructions for the execution processor POP. If all operators are diadic ones, then $M = N-1$, but in general we have $M \geq N-1$. Hence the initial number of orders is equal to $M+N \geq 2N-1$.

Each time a new group is entered, one must jump over its sequence of variables, that is to say that the number of extra UP orders to be generated is equal to the number p of groups in the input string. The evaluation of the number of extra DOWN orders is a bit more difficult to do. Let k_i be the number of extra DOWN orders generated during the execution of group G_i. The maximum value of k_i is equal to $i-1$, since the queue has a maximum number of holes equal to $i-1$. Furthermore, each executed DOWN order decreases the number of further possible DOWN orders by 1, since one hole is suppressed.

Hence we have $O \leq k_i \leq (i-1) - \sum_{j=1}^{i-1} k_j$, for all i.

When adding the above formula for all i, we get:

$$O \leq \sum_{j=1}^{p} k_j \leq \sum_{j=1}^{p} (j-1) - \left[\sum_{j=1}^{i} k_j + \ldots + \sum_{j=1}^{p-1} k_j \right]$$

which becomes:

$$\sum_{i=1}^{p} \left(\sum_{j=1}^{i} k_j \right) \leq \sum_{j=1}^{p} (j-1) = \frac{p(p-1)}{2}.$$

Hence we have the number of extra Down orders given by

$$O \leq \sum_{j=1}^{p} k_j \leq \frac{p(p-1)}{2} - \frac{(p-1)(p-2)}{2} = p-1.$$

However, if the number N of variables is less than 2p ($N \leq 2p$), than the maximum number of diadic operators is less than $2p-1$. As the p operators O_1^1, $O_2^1 \ldots$, O_p^1 cannot imply the generation of a DOWN order, we have only $p-1$ operators which can do so. Hence the maximum number of extra DOWN orders is equal either to

$$N-p-1 \text{ if } p+1 \leq N \leq 2p, \text{ or to } p-1 \text{ if } N \geq 2p.$$

Now it is easy to see that the minimum rate of extra-orders is given by:

$$\text{MINRATE} \leq \frac{P}{2N-1}$$

and the maximum is given by:

$$\text{MAXRATE} \leq \frac{N-1}{2N-1} \text{ if } p+1 \leq N \leq 2p,$$

$$\text{MAXRATE} \leq \frac{2p-1}{2N-1} \text{ if } N \geq 2p$$

Example:

$$S = (A, B, C, *, +, D, -)$$

In this case, there are $p=2$ groups, and $N=4$ variables. Hence $N \geq 2p$

$$\text{MINRATE} \leq \frac{2}{7}$$

$$\text{MAXRATE} \leq \frac{3}{7}$$

Trade off between compile time and execution time for the generation of the extra-orders

The generation of the input string is performed by the compiler, which could easily generate extra-orders. However, the compiler becomes machine dependent in such a case, and the size of generated code is increased, as the number of instruction fetches from main memory at execution time. Hence, the best solution consists in the analysis of the polish string at execution time, since the compile time solution would slow down the global performance of the machine. The generation is made by the analysis processor PAN, which can be considered as the control processor, and execute all the control function of the computer.

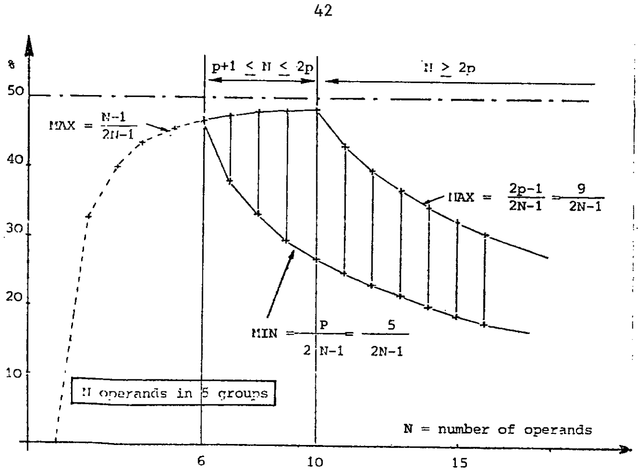


Figure 8

Algorithm for the generation of extra-orders at execution time

Generation of orders is performed by the analysis processor PAN. Given the input polish string, this processor must generate orders towards both processors PAC and POP, using the theoric state of the queue, represented by the sequence $\{(d_i, t_i)\}$ defined earlier. The sequence $\{(d_i, t_i)\}$ can be managed using a push-down control stack. Let TS and STS be the two topmost elements of the stack, which respectively hold the couples (d_k, t_k) and (d_{k-1}, t_{k-1}). These two variables TS and STS are structured as two fields called D and T, so d_k is equivalent to TS.D, t_k to TS.T etc.

The generation algorithm is illustrated in Figure 9, where the symbol (F_i) represents the name of the function to be executed when the next symbol in the input string defines the state transitions. The next symbol type is represented either by [variable] or by [operator].

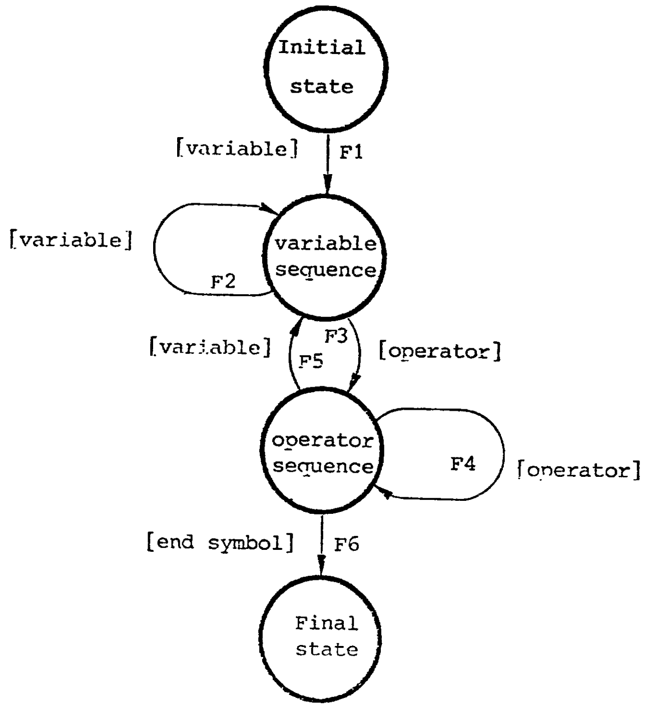


Figure 9

Function F1 initializes the queue state, by pushing the couple (−1,O) onto the control stack.

Function F2 generates an Access Order towards the Access Processor, next counts the number of variables in the current group, by incrementing the top of stack

$$(\text{TS.D} \leftarrow \text{TS.D}+1)$$

Function F3 is executed when the first operator of the current group is encountered. It occurs on the transition from $V_i^{n_i}$ to O_i^1. Its function is the generation of an UP extra-order, with a parameter n equal to the number of variables in the current group, which has been evaluated by function F2 (the parameter is equal to n_i since F2 has been executed n_i times).

Function F4 generates an Operation order towards the execution processor POP. If the operator is a diadic one, then the queue state is updated, modifying the top of stack element: TS.D←TS.D−1 since there is one less operand in the last group, and TS.T ←TS.T+1 since there is one more empty location in the last group.

If TS.D becomes zero, an extra-order DOWN is generated, with a parameter n equal to the second top of stack element STS: one generates DOWN (STS.T). Next the queue state is modified:

STS.T÷STS.T+TS.T since the length of the hole must be incremented and PULL(TS) since the topmost element is deleted (STS becomes the top of stack).

Function F5 is executed on the transition from $O_i^{m_i}$ to V_{i+1}^1, i.e. when the first variable of a new group is encountered. This function initializes a new element on the stack, only if the previous top of stack represents a real group.

if TS.D>O then PUSH(1,O)
else (TS.D←1, TS.T←O).

Function F6 is executed on the occurrence of the end-symbol, which is any separator symbol between two expressions (e.g. an Assign Symbol). It only verifies the correctness of the control stack state, which must be the empty state.

ARCHITECTURE OF A HIGH-LEVEL PIPELINE COMPUTER

The first section of this paper has shown that pipeline execution of polish string code is made possible using a FI-FO queue.

In this section, an architecture is presented for a high-level pipeline computer whose code is in polish string format. The dependency problem is first studied and a solution is given.

THE DEPENDENCY PROBLEM

Suppose, for example, that the high level instruction "X←<exp>" has been prepared in both PAC and POP instruction queues, or is in the process of execution by processor POP, when the access to variable X instruction enters the access processor PAC. The access processor must be able to detect the fact that both instructions refer to the same variable X, and that the second instruction might have to be deferred until the completion of the first instruction, since a reference to the location of X in main memory would not give the true value of variable X, but its old value.

The solution consists in the definition of a content addressable memory, organized in a FI-FO mode, which holds the name of the variables whose modification is deferred, and the name of the variables which have been just modified. In the first case (deferred modification), any reference to the variable is processed as an indirect reference by creating a link between the assignment and all the deferred references. In the second case a reference to main memory is eliminated, since the current value of the variable is available in the content-addressable memory after the completion of the last assignment.

Using the above mechanism, the access instruction (VALUE X) can be deferred until the completion of the assignment. All the deferred references are linked together, eliminating a number of memory references equal to the number of linked locations.

THE CONDITIONAL BRANCH PROBLEM

Both PAC and POP processors may be considered as SLAVES of the PAN processor in the following sense: then only execute the internal instructions that they receive from the PAN processor. Moreover, every in-

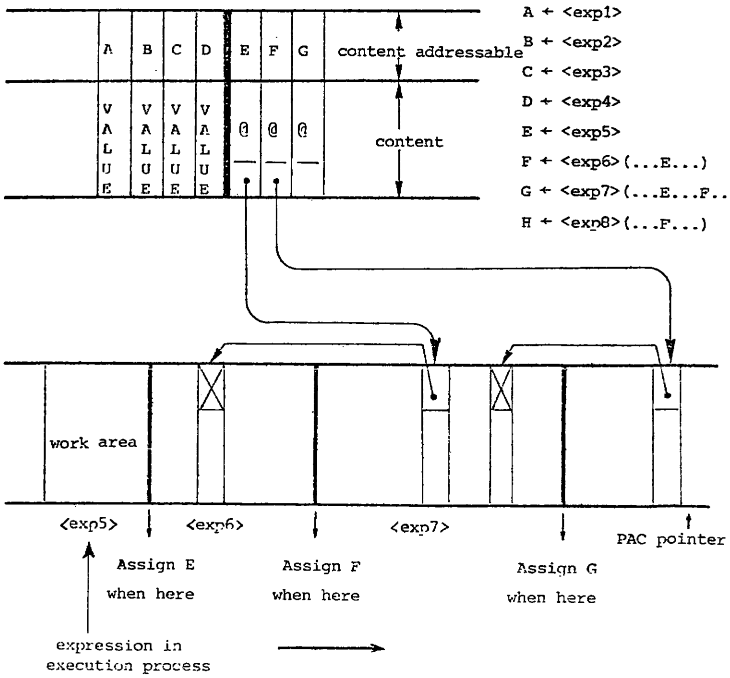


Figure 10

struction belonging to the input string is fetched by the PAN processor. So, this processor can be considered as the MASTER of the control, and it is involved in all control functions in the computer during execution of a single high-level program.

When a conditional branch occurs, the PAN processor is not able to fetch the next instruction, since the conditional expression is currently in the process of evaluation. However, the PAN processor may *choose* one instruction among all the possible next instructions (generally two). The probability of a bad choice strongly depends on the context of the conditional branch: it is much lower for a LOOP statement than for an IF statement.

When a choice is made, we say that the PAN processor enters a Conditional State, characterized by the fact that its activity is limited to a preparation work. Especially, if a conditional branch occurs during this conditional state, no choice is made, the processor waiting for the resolution of the first conditional branch.

When the value of the conditional expression is available, two cases may occur: either the choice was good, in which case the process goes on without any modification, or the choice was bad, in which case all the prepared work must be disabled. This is simply achieved by writing as "empty" the input instruction queues of both PAC and POP processors which hold bad instructions and updating both evaluation and dependency queues by deleting the sequence of "bad" operands or "bad" deferred variables (they are "bad" because they belong to the bad choice).

HOW TO SAVE THE EVALUATION CONTEXT

The evaluation context (intermediate state of the evaluation queue) must be saved when a "function call" occurs within an expression. Function calls are introduced in the input polish code in the following manner: any operand can be either a variable name or a function call.

The syntax of a function call is

(FCALL <name>, <parameter-list>, ENTER).

When FCALL is decoded by the analysis processor PAN, a special order is sent to the PAC instruction queue that calls for the address of a save area. This area is allocated on the top of a push-down stack, since several function calls can be nested. Next, processor PAN, which knows the current state of the evaluation queue, generates a sequence of DOWN and SAVE orders towards the POP instruction queue. Hence the current state of the evaluation queue can be saved before the function is entered, all previous results being compacted into the save area.

When the function is returned, processor PAC is capable to restore the initial state, pushing the function result just after the restored values, and the evaluation process goes on.

THE GLOBAL ARCHITECTURE OF THE COMPUTER

The pipeline computer architecture is shown by Figure 11. Each of the three processors can run concurrently. Their synchronization is data-driven. Processor PAN fetches high-level polish form instruction stream from Main Memory, and executes all the control instructions (IF, LOOP, GOTO, . . .).

It analyzes input expressions and generates two internal instruction streams towards both PAC (access instructions) and POP (execution of operation).

Two synchronous exchanges occur: the first one is concerned with the transmission of the entry address when a procedure is entered. The second one is the completion of a conditional expression which can disable the choice made by processor PAN on the occurrence of a conditional branch.

SUMMARY

The pipeline architecture described in this paper is potentially capable of high performance. Its input code is in a polish string format that can be directly translated from a block structured high-level language. The pipeline execution is based on a natural decomposition into control, access to operands and execution of operations, executed by three concurrent processors.

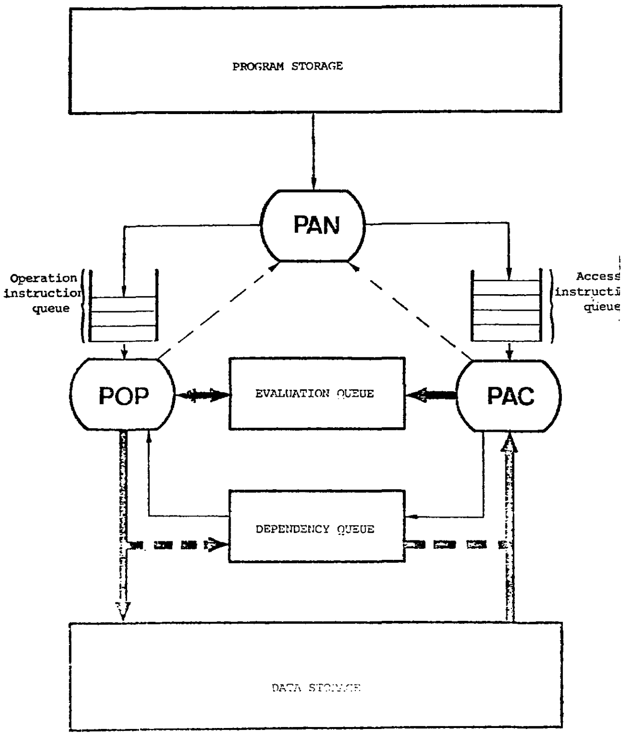


Figure 11—Architecture of the pipeline computer

This computer architecture is currently in application for the design of a PASCAL computer. Each of the three processors is in design using high speed macrologic components in Low Power Schottky Technology.[11,12]

ACKNOWLEDGMENTS

The authors would like to thank Dr. F. Anceau for his constant encouragement and helpful discussions and all members of the Computer Architecture Group of the University of Grenoble for their constructive criticism. Special thanks go to Mrs. H. Diaz for preparing the paper.

REFERENCES

1. Anderson, D. W., F. J. Sparacio and R. M. Tomasulo, "System/360 Model 91: Machine Philosophy and Instruction Handling," *IBMJR&D,* 11, No. 1, January 1967, pp. 8-24.
2. Thornton, J. E., "Parallel Operation in the Control Data 6600," *AFIPS FJCC Conf. Proc.* 26, Part II, Washington, D.C., Spartan Books, 1964, pp. 33-40.
3. Nelson, H. L., *Program Optimizing Techniques for the CDC 6600 Central Processor,* UCRL-12489, University of California, Lawrence Rad. Lab. 1965, also NASA N66-20536.
4. Squire, J. S., "A Translation Algorithm for a Multiple Processor Computer," *Proc. 18th ACM Nat. Conf.,* Denver, Colorado, 1963.
5. Baer, J. L. and D. P. Bovet, "Compilation of Arithmetic Expressions for Parallel Computations," *IFIPS Congress 68,* August 1968, pp. B4-B10.
6. *BURROUGHS B5500 Information Processing Systems Refence Manual,* Burroughs Corp., Detroit, Michigan, 1964.
7. Stone, H. S., "A Pipeline Push-down Stack Computer," *Parallel Processor Systems, Technologies and Applications,* Chapter 12, pp. 235-249.
8. Abrams, *An APL Machine,* Stanford University, No. CS 70158, 1970.
9. Tomasulo, R. M., "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," *IBM Journal,* January 1967.
10. Flynn, M. J., "Very High Speed Computing Systems," Proceedings of the IEEE, Vol. 54, No. 12, December 1966.
11. Baille, G. G. and J. P. Schoellkopf, "Evaluation of a Polish Form Expression on a FI-FO Queue," *1975 SAGAMORE Computer Conference on Parallel Processing,* August 19-22, 1975, p. 200.
12. Anceau, F., G. G. Baille and J. P. Schoellkopf, "Conception Descendante des Machines Informatiques, Application à Une Machine PASCAL," Rapport Contrat IRIA-SESORI No. 74-156, October 10, 1975.

Evolution of computer memory structure*

by YAOHAN CHU
University of Maryland
College Park, Maryland

ABSTRACT

The memory structure of a computer refers to those hardware elements that store the data elements and the related information during program execution. It has a great impact upon the usage and the programming of the computer. Since the advent of commercial electronic digital computers, the memory has made great hardware advances in speed, capacity, size, cost and reliability. It has also made great organizational changes.

This paper reviews the memory structure of typical computers from the viewpoint of the software need. It emphasizes the structural advances in satisfying the software need of various types of data elements and data structures. It presents observations of the evolutionary nature of the memory structure development toward the high-level language architecture.

MEMORY STRUCTURE OF VON NEUMANN MACHINES

Von Neumann machines refer to those computers which make use of a random-access memory (often known as the main memory), in which the program and data are stored. One unique characteristic is that the program is a sequence of instructions, and each instruction typically has an op-code to represent an operator and an address to locate an operand in the main memory. Another unique characteristic is that there is no distinction between the program and the data that are stored in the main memory; thus, an instruction can be treated as the datum and vice versa. Most of the electronic digital computers that have been built are Von Neumann machines.

IBM 7090/7094 computers

The IBM 7090/7094 family of computers were among the most popular large-scale computer systems during the period of the early 1960's. The memory structure, shown in the block diagram of Figure 1, is quite simple. It consists of a main memory with a capacity of 32K 36-bit words, and 3 or 7 index registers. The structure recognizes only binary numbers; integers and real numbers have different formats. It did not recognize characters, strings, stacks, and the like; they must be interpreted by means of subroutines or program segments. It is apparent that the memory structure is rather rudimentary for high-level programming languages.

IBM 360/370 computers

Since 1964, IBM introduced System/360 family and later in 1969 System/370 family of computers.[1] The memory structure, shown in the block diagram of Figure 2, consists of a main memory, 16 general registers (each of 4 bytes), and four floating-point registers (each of 8 bytes). The main memory can be expanded to a maximum of about 16-million bytes. It can recognize binary and decimal numbers, strings of a varying number of characters, and strings of a fixed number of bits. There are four formats for binary numbers, two formats for decimal numbers, one format for a string of bits and one format for a string of characters. There are four kinds of operands that can be addressed: the register operand, the storage operand, the

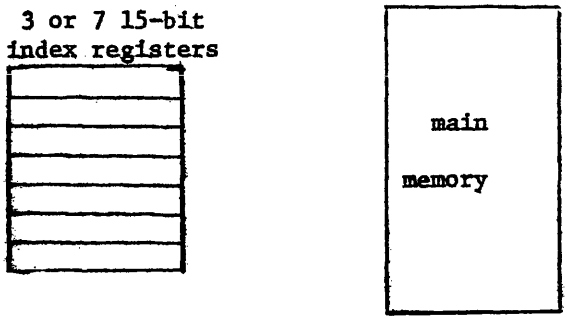


Figure 1—Memory structure of the IBM 7090/7094 computers

* This research is supported by Grant NSF DCR75-05505 from the National Science Foundation to the Department of Computer Science of The University of Maryland.

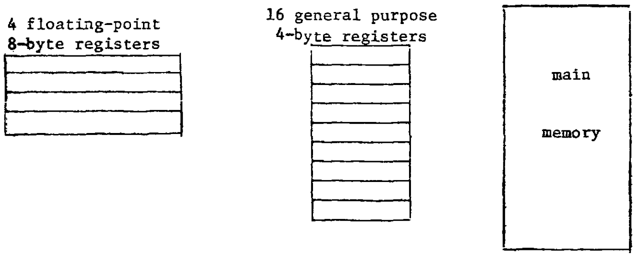


Figure 2—Memory structure of the IBM S/360 and S/370 computers

immediate operand and the string operand. It is obvious that the memory structure is more sophisticated than those of the IBM 7090/7094 family of computers, but it is still far from adequate for high-level programming languages.

IBM S/360-85 system was the first to have the so-called "cache memory."[13] It was a buffer memory installed between the main memory and the CPU. It was a very-high-speed semiconductor memory, whose presence was transparent to the programmer. The use of the very high-speed cache memory can greatly improve the speed of computer operation without increasing the speed of main memory operation. Thus the use of a buffer memory enhances the memory speed by means of memory organization.

The current IBM S/370 family of computers which succeeds the IBM S/360 family extensively provides the virtual memory. A virtual memory makes the main memory appear to have an enormous capacity to the programmer. It has been implemented by a combination of hardware and software to appear as a one-level memory. Thus, the use of one-level memory enhances the memory capacity by means of memory organization. The virtual memory was first made commercially available in Burroughs B5500 system early 1960's.[11] A hardware virtual memory (virtual memory becomes a misnomer) was demonstrated in the Symbol system early 1970's.[12]

MEMORY STRUCTURE OF STACK-ORIENTED MACHINES

Stack-oriented machines are those machines where there is a hardware stack. This stack is provided to perform only some of the functions in program execution control, data handling and referencing, and expression evaluation. For this reason, it is called a stack-oriented machine. It is a step in memory structure evolution. It is an intermediate between a Von Neumann machine and a stack machine. The memory structures of Intel 8008 and 8080, the DEC PDP 11/45 and the Burroughs B1700 systems are introduced below.

Intel 8008 and 8080 structures

The Intel MCS-8 and MCS-80 microcomputer systems are stack-oriented machines. The hardware stack is used only for subroutine return linkage and temporary operand storage. It is not used at all for arithmetic expression evaluation.

The Intel MCS-8 microcomputer system makes use of an Intel 8008 microprocessor.[9] It executes a typical instruction in 12.5 microseconds, and recognizes only an 8-bit binary number. The memory structure is shown in Figure 3. It consists of a main memory expandable to 16K bytes, an address stack of eight 14-bit registers and seven 8-bit registers. The address stack stores the return addresses of subroutine calls, which is transparent to the programmer. One of the seven registers is the accumulator; nearly all arithmetic and I/O operations use it. Because the 8008 does not have instructions that have direct addresses, two of the remaining six registers are used for all references to main storage. In order to fetch an operand from storage, the program must load an address into two specific registers. This means that at least three instructions must be used to refer to the arbitrarily placed data, this is a major disadvantage.

The Intel MCS-80 microcomputer system[10] is an outgrowth of the MCS-8. It makes use of an 8080 microprocessor. It executes a typical instruction in 2 microseconds. It recognizes only an 8-bit binary number and has only addition and subtraction instructions. The 8080 microprocessor offers 30 additional instructions, a speed improvement of at least 10 times, and possible code reduction of up to 30 percent. The memory structure shown in Figure 4 consists of a main memory with a maximum capacity of 64K bytes, six 16-bit registers, and an accumulator. Of the six registers, one is a stack

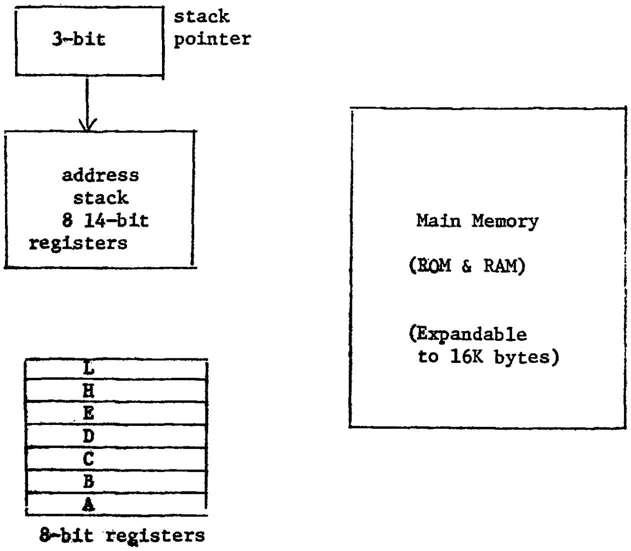


Figure 3—Memory structure of the Intel MCS-8 system

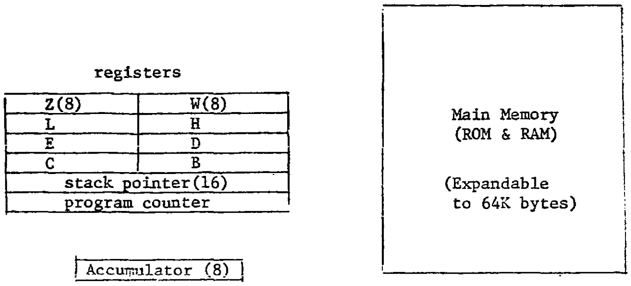


Figure 4—Memory structure of Intel MCS-80 system

pointer. The stack pointer is used to place all return addresses in the stack which is in the RAM. Instructions are available for directly handling the stack. Another of the six registers is the program counter. The Z and W registers are for the internal use of the microprocessor. The remaining three 16-bit registers can also be addressed as eight-bit registers.

The 8080 allows no index addressing, but it has instructions that permit explicit addressing of storage locations. Where the 8008 only allows a single pair of eight-bit registers to address the memory, the 8080 permits any of the three main registers to hold and output an address when using register-indirect addressing.

DEC PDP 11/45

The DEC PDP 11/45 is a stack-oriented minicomputer system.[8] It accepts a conventional machine-level language. Although it provides optional hardware for memory segmentation, its facility for the hardware stack is rather limited.

Memory structure

The memory structure of a PDP 11/45 with the memory segmentation hardware is shown in Figure 5.

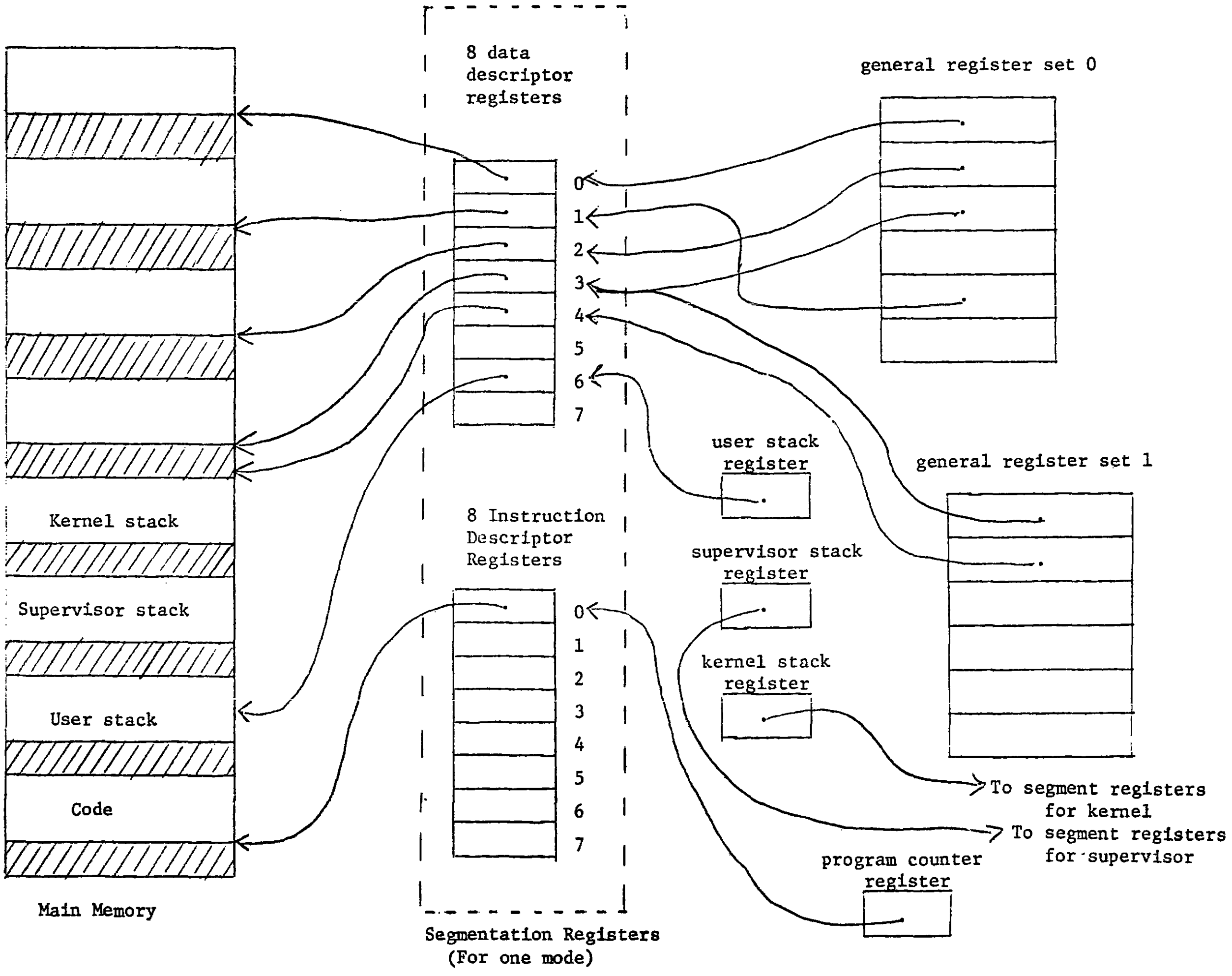


Figure 5—Memory structure of the DEC PDP 11/45

There are two sets of general purpose registers; each set has six 16-bit registers. Only one set is active at a time. Each set can be used to store either two 8-bit bytes, a 16-bit binary number, or a pointer to a segmentation register. There is also a program counter which points to the code currently being executed.

There are three modes (or states) in which the hardware may run: the kernel, the supervisor, and the user modes. There are three sets of segmentation registers; each set has 16 32-bit registers. Each set is for each mode; there is a total of 48 registers. Each 16-register set can be split into two sets of 8 registers: 8 Data Space Descriptor Registers and 8 Instruction Space Descriptor Registers. There is also one hardware utilized top-of-stack pointer register for each mode.

Hardware recognized data elements

The data elements that the hardware can recognize directly are a one- or two-byte string, or a 16-bit binary number.

Recognized data structure

The only data structure it recognizes is a stack which is in the main memory. The stack is used by the hardware only for subroutine and interrupt return linkage.

Referencing

The PDP 11/45 is interesting in that it is a byte or a word (2 bytes) addressable machine. There are two classes of instructions (including two types of load and store); one is byte-oriented and the other is word-oriented. The hardware stack can also be referenced by the software by the use of a powerful auto-increment and auto-decrement class of instructions.

All data elements and structures in the main memory are referenced indirectly through one of the 16 active segmentation registers. The referenced segmentation register contains a special pointer to an area in main memory. This pointer is called a descriptor. It not only points to the address of the area but also specifies other information about the area. A segmentation register descriptor basically specifies the main memory address of an area and its length, whether the area is stack or not or whether the area is read-only or read-write.

The stack (which is in main memory) is used by the hardware only for subroutine and interrupt return linkage.

Burroughs B1700 structure

The Burroughs B1700 is a recently available small-scale micro-programmable computer.[4] Like the Intel 8008 and 8080, it is a stack-oriented machine. The major innovation of the B1700 is the use of a writable control memory to store the microprograms. The machine-level language is an assembly-level but syntax-oriented language called S language. A high-level language source program is translated by a compiler into a S language code which is interpreted by a matched microprogram. There can be more than one pair of a S language and its matched microprogram.

Memory structure of the B1700

Figure 6 shows the memory structure. There are 6 general-purpose registers, a hardware stack for subroutine return linkages and for general operand storage, a control memory for microprogram, a register table to hold either base registers or general operands, register F to reference the contents of the main memory itself. Register MAXS specifies the maximum size of the main memory. Also shown in Figure 6 are two registers, BR (base register) and LR (limit register); they are used to mark the start and the end of the data area in the main memory of the current program. All data accesses made by the program are checked by the hardware to insure that they do not exceed these bounds.

The microprogram is stored in the control memory, but it can be extended into the main memory. Register MBR points to where the microprogram code overflow starts in the main memory. Register A is the microprogram instruction counter. The read-only register MAXM stores the available control memory. Register TOPM points to the end of the microprogram in the control memory.

Hardware recognized data elements

The B1700 hardware is capable of recognizing variable length binary numbers (1 to 24 bits in length), variable length decimal numbers (1 to 6 digits in length), and variable length strings (1 to 3 characters in length) explicitly. It can recognize any of these elements in longer lengths but the use of more than one micro-instruction is needed.

Hardware recognized data structures

Aside from the either 16 element (B1714 and B1716) or 32 element (1724 and B1726) hardware stack, the B1700 memory structure recognizes no data structures.

Referencing

Any data element in a register is referenced directly by the register name. Certain registers, as shown in Figure 6, have addressable fixed subfields. Register T

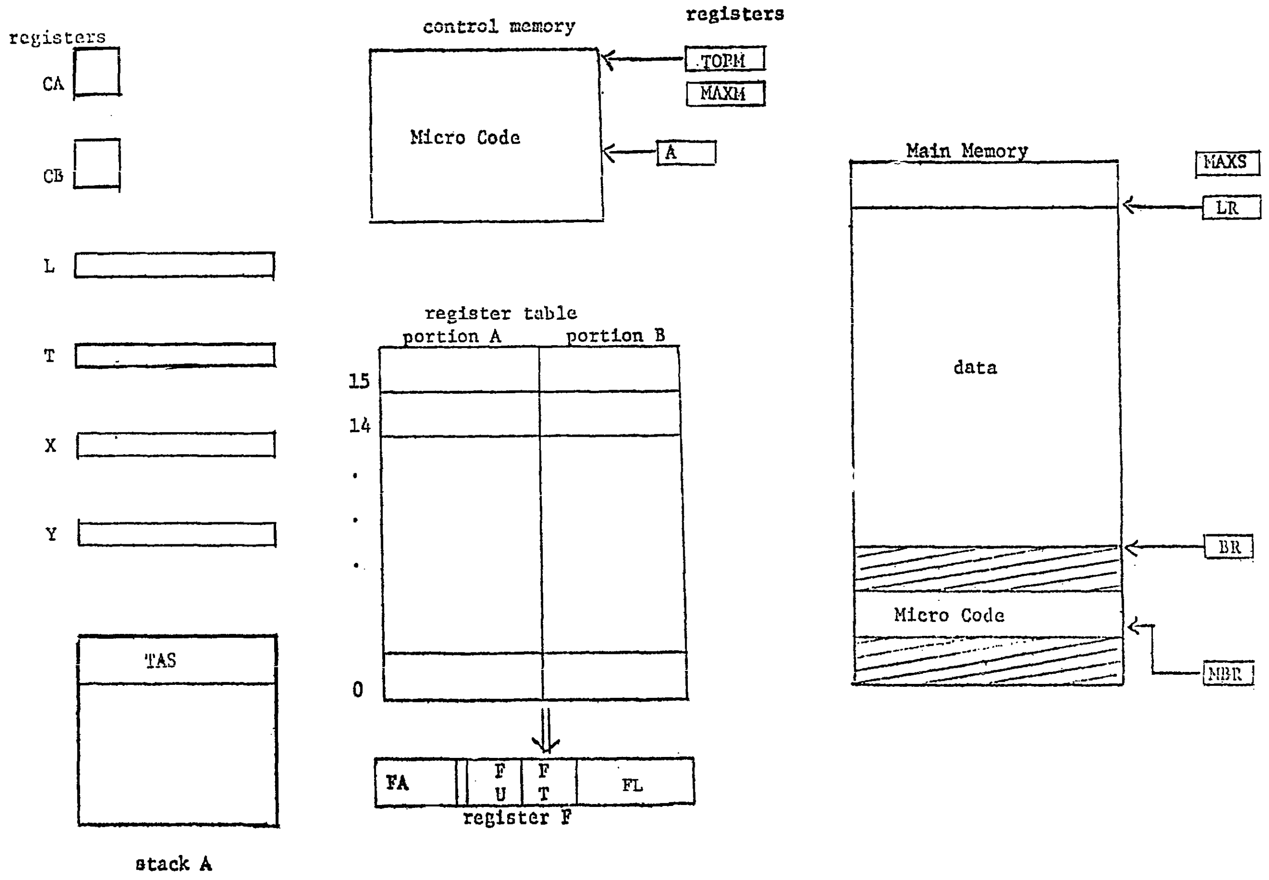


Figure 6—Memory structure of the Burroughs B1700 system

is also provided with special hardware so that any field of this register may be accessed. A data element in main memory is accessed by placing a data descriptor in register F. The format of this data descriptor, identical to that of register F shown in Figure 6, has four basic fields: FA (Field address), FU (Field unit), FT (Field type), and FL (Field length). Micro-instructions are available to transfer any one of these fields to some registers. Field FA contains the absolute bit address of the data area. Field FU may contain the number of bits per byte in the data item (e.g., binary is 1 bit/byte, octal is 3 bits/byte, etc.). Field FT may hold data type. Field FL may hold the length of the data area.

MEMORY STRUCTURE OF STACK MACHINES

Stack machines are those machines which have a hardware stack. The stack is particularly necessary for machines whose machine-level languages have a high-level syntax such as the polish notation. Burroughs B5500 system was the first stack machine.

In a stack machine, the hardware stack can push down or pop up an operand. It can test for stack overflow or underflow. It can be used for expression evaluation. It can store the return address and pass parameters of a procedure call. It can handle the interrupts. In fact, all operands go through the stack.

This section briefly describes the memory structure of the family of Burroughs B5500, B6700, and B7700 stack machines.

Burroughs B5500 structure

Burroughs achieved a great stride in computer architecture through the introduction of the B5500 system in early 1960.[2,3] This system was specifically designed for the execution of a high-level language called Extended Algol which is a dialect of Algol 60.

It is important to note that the philosophy of the design of the B5500 was to eliminate the assembly language programming. Aside from some rather rare occurrences within the operating system, this goal has essentially been achieved.

The B5500, as well as the B6700 to be mentioned later, is a stack machine. The stack is not only used for subroutine and interrupt return linkage but also for arithmetic evaluation and procedure parameter storage.

Memory structure of the B5500

The memory structure of the B5500 is sketched in Figure 7. There is a main memory with a capacity of 32K of 48-bit words, two registers S and R pointing to the main memory, two top-of-stack buffer registers A and B, two registers G and H which indicate a bit position within the A register, and two registers K and V which indicate a bit position within the B register. The pairs of G, H and K, V registers allow accessing any contiguous string of bits in either the A or B registers.

Hardware recognized data elements

The hardware can directly recognize and manipulate binary and decimal numbers, strings of a varying number of characters, and strings of a varying number of bits. There are two formats of binary numbers (real-single-precision and real-double-precision), one format of decimal numbers, and one format of character string, and one format of bit strings.

Hardware recognized data structures

The B5500 hardware directly interprets stack, single dimensional array structures, and singly linked lists.

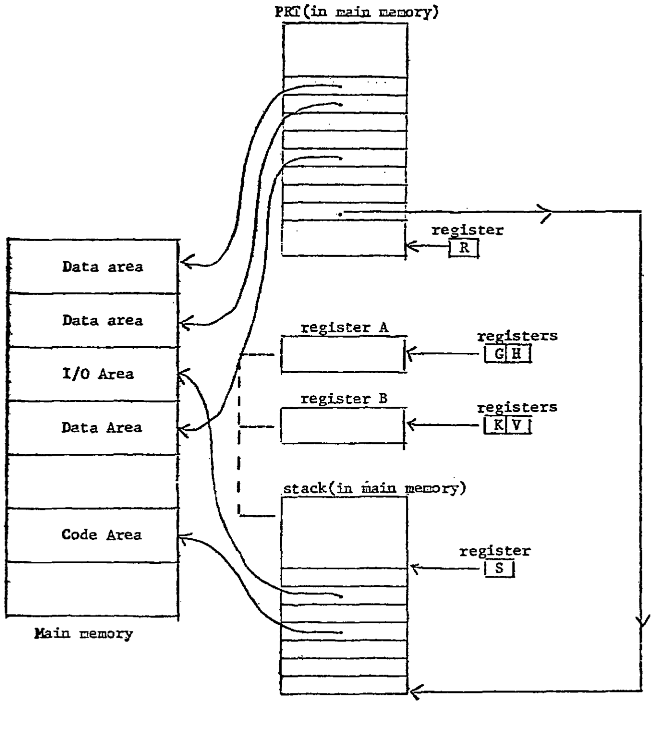


Figure 7—Memory structure of the Burroughs B5500 system

Referencing the data structures and data elements

Each program in execution has a PRT (Program Reference Table) which contains all scalar data elements and pointers to all arrays in addition to a hardware maintained stack. As shown in Figure 7 register R points to the base of the PRT of the currently executing program. Registers A and B contain the top two elements of the hardware maintained stack, while register S points to the top stack element in the remainder of the stack which is in main memory.

The B5500 has four different types of addressing it can perform: (a) R-relative (address an element of the PRT), (b) top-of-stack (e.g., multiply top two elements of stack), (c) stack relative address (a procedure programmer), and (d) absolute addressing.

R-relative addressing allows the accessing of binary numbers and bit strings less than 48 bits. in length. Top-of-stack addressing allows the use of the stack for evaluating arithmetic expressions and is also used for all temporary workspace (e.g. index operations are performed in the stack).

In order to access the other data types of decimal numbers, character strings, bit strings of 48 or more bits, and arrays, a combination of these addressing types is used. Each datum in main memory which is not merely a binary number or a 0-47 bit bit-string is pointed to by a special pointer in the PRT. This pointer is called a *data descriptor*. The data descriptor has 3 hardware recognized fields: P, WC, and ADDRESS. The one-bit P field indicates if the descriptor points to an area in main memory or on the disk memory. The VC field specifies the size of the area in 48-bit words (or 8 characters). The ADDRESS specifies the location of the data area either in the main memory or disk (depending on field P).

In order to reference a data item described by a data descriptor, an index (if needed) and the descriptor (via R relative addressing) are first pushed onto the stack. Next, if an index was specified, the hardware checks to see if it is within the size of the WC field. If it is, the index is added into the ADDRESS field of the descriptor to give the absolute address of the data item.

The descriptor can be used to either load or store a value into the item referenced. For example, to reference A[3], the value of the index, 3, is placed on the stack and a "value call" operator on the address of A is performed by the software. The hardware automatically: (a) add the index to the array's descriptor, (b) make sure the subscript is in bounds, (c) delete the index from the stack, and (d) place the addressed array element on the stack.

There is also a special instruction, LINKED-LIST LOOKUP, which searches a singly-linked list. This list is created by software.

Burroughs B6700/7700

The Burroughs B6700 and B7700 systems are basically large-scale, improved-versions of the B5500 system, commercially becoming available almost 15 years ago. Rather than indicating that the architecture of the B6700 and B7700 is antiquated, the fact is the architecture of the original B5000 and B5500 was greatly in advance of the time. The Burroughs B6700/7700 systems are perhaps the first stack machines, where all data storage other than arrays is actually performed by the powerful stack organization.

Memory structure of the B6700/7700

The memory structure of the B6700/7700 family of computers[6,7,14] is sketched in Figure 8. As shown, registers A, X and B, Y store the top two elements of the current stack (four registers for two double precision operands). Registers BOS, S, and LOS point to the base, the current top, and the limit of the current stack in main memory, respectively. There is a software-maintained but hardware-accessed Stack Vector Table in main memory. The Stack Vector Table stores one pointer to the base of each stack in main memory. The table entry for the current stack is pointed by register SNR.

Also shown, there is a 32-entry register-table D which is used for addressing and an associated register LL which specifies the highest register in the D table which is valid. There are 3 "vector mode" registers for storing Pointer A, Pointer B, and Pointer C; these are index registers for efficient one-dimensional array accessing.

Hardware recognized data elements

The B6700/7700 hardware recognizes by means of data descriptors binary numbers, strings of a variable number of 1 to 48 bits, decimal numbers, strings of a variable number of characters (either 6 or 8 bits). There are two formats of binary numbers: single-precision real and double-precision real. There is one format of all other types of data elements.

Hardware recognized data structures

The B6700/7700 hardware recognizes stacks, multidimensional arrays, and singly linked lists.

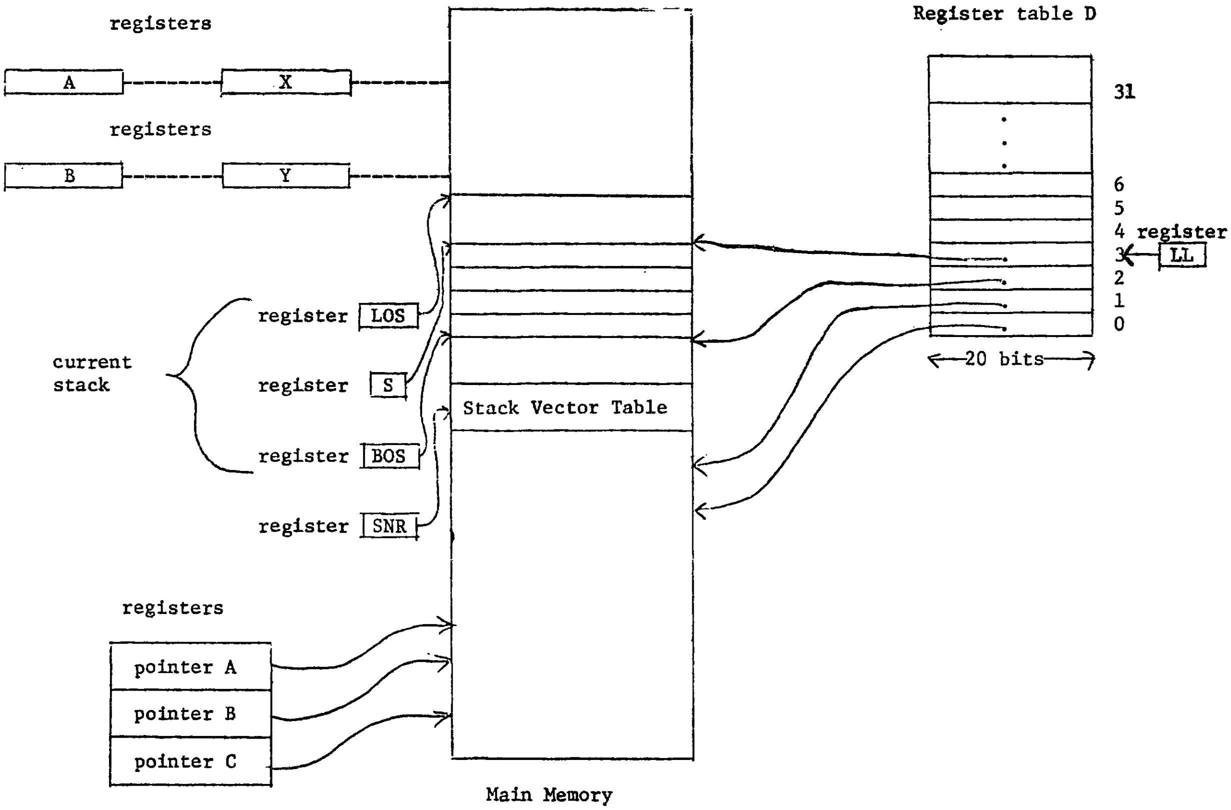


Figure 8—Memory structure of the Burroughs B6700 and B7700 computers

Referencing

Every memory word has a 3-bit tag. To address a memory word, the hardware has three different addressing schemes which depend on whether the memory word to be addressed is in the current stack, in another stack, or external to all of the stacks (e.g., an array or a string). Figure 9 illustrates the stack addressing scheme. As shown, for each block level of the sample Algol program, there is a corresponding entry in the D register table which points to a special set of control words (i.e., MKSW and RCW) for that block. Above these control words are stored the declared simple variables for the block.

Representation of a three-dimensional array in the memory structure is illustrated in Figure 10. The array is structured as a tree with the first level having the array descriptor, the second and third levels having dope vectors of data descriptors, and the fourth level being actual array elements. To address array element A[1,1,1], each index is placed onto the stack and then a "value call" is performed on array A by software. The hardware automatically follows the data descriptors as indicated by the indices on the stack. It automatically deletes the indices and places the correct array element on top of the stack. It automatically checks the number of subscripts to see whether they are in bounds when the reference is made using the contents of the data descriptor dope vectors.

The B6700 (optionally)and the B7700 (standard) have three hardware index registers which are used in a special hardware "vector mode" to allow very efficient access to single-dimensional arrays.

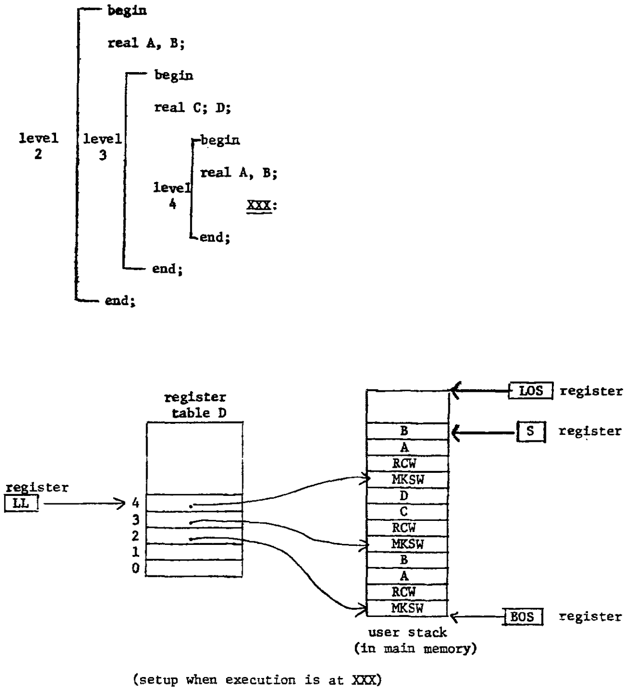


Figure 9—Stack addressing scheme in the B6700/7700

The hardware references the stack by using the hardware top-of-stack pointer, register S.

The hardware has a special linked-list search operator which will follow a software-constructed singly-linked list until the current mode has a value greater than or equal to a specified value.

In addition, the B6700/7700 systems have an interesting hardware operator which serially searches a data area from the end to the beginning for a data element identical to the given argument. This operator allows an easy software simulation of an associative memory.

Speed improvements in B7700

The Burroughs B7700 memory[7] has some major differences from the B6700 memory: a clock rate of 3 times as fast, extensive incorporation of fault detection circuitry, single-bit error correction in memory, 4-way interleaving of the main memory, sophisticated I/O operations, and the use of an associative memory.

The most interesting aspects of the B7700 memory structure is the use of a high-speed semiconductor memory and an associative memory to assist in stack handling. The high-speed semiconductor memory is basically used as a cache memory to hold the top words of the current stack. This memory can hold a maximum of 32 words, which can be operands, data descriptors, code descriptors, etc. The 32 51-bit word associative memory contains the most frequently-used operands and descriptors in the stack which are not in the semiconductor memory but can be any other words of the stack. Each entry in the associative memory essentially has two fields: a 20-bit address indicating the memory location of the original operand or descriptor, and a 51-bit copy of the operand or descriptor.

When any memory reference is made on the B7700, the storage control unit first checks the semiconductor memory and associative memory for the data item. If it is found and if it is a read operation, the reference is complete. If it is found but if it is a write operation, the word is overwritten in the semiconductor memory or associative memory. A write request to the main memory is hardware-queued for the memory controller so that the original item in the main memory is also updated.

MEMORY STRUCTURE OF THE SYMBOL SYSTEM

An historical event in computer architecture occurred during the 1971 Spring Joint Computer Conference. Rex Rice and others[12] of Fairchild presented a series of papers which described the design and con-

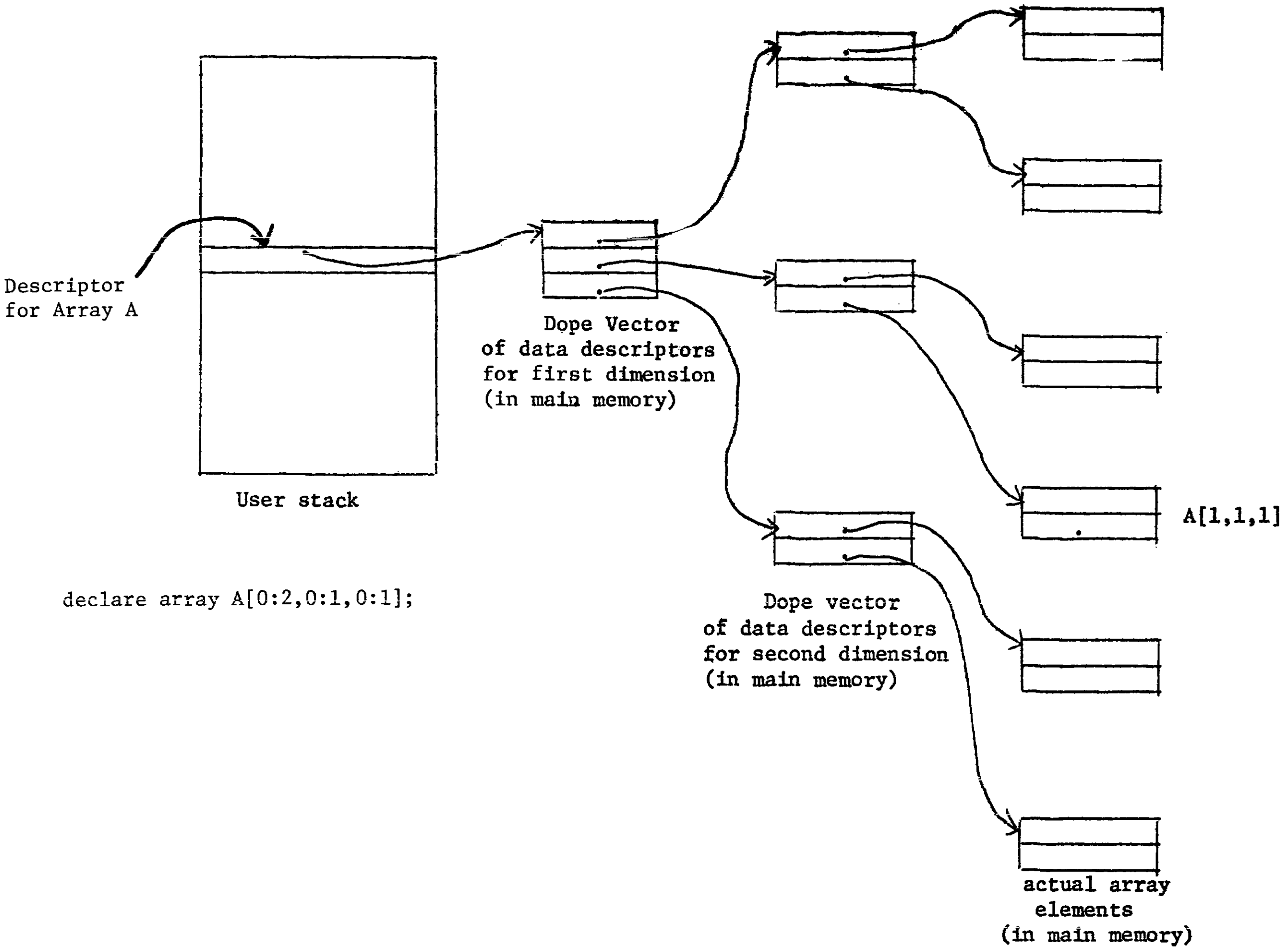

Descriptor for Array A
User stack
declare array A[0:2,0:1,0:1];
Dope Vector of data descriptors for first dimension (in main memory)
Dope vector of data descriptors for second dimension (in main memory)
A[1,1,1]
actual array elements (in main memory)

Figure 10—Representation of an array in main memory

struction of the Symbol computer system. The Symbol system has a high-level Algol-like machine language, called Symbol, which is additionally capable of describing variable length data and requires no type and size declarations (as conversion and memory allocation are handled automatically). It has no conventional set of instructions. It directly accepts programs written in the Symbol language. It is a functionally-organized multi-processor system and designed for multiple access by terminals. In addition to an arithmetic processor, a channel controller, and a disk controller, there are a hardware translator, a hardware text editor, a hardware format processor, a hardware reference processor, a hardware system supervisor, and hardware virtual memory. There is very little software. The Symbol system is now being evaluated at Iowa State University under the sponsorship of the U.S. National Science Foundation. This project has undoubtedly demonstrated the feasibility of a computer system whose machine language is a high-level language.

Hardware virtual memory

The memory structure consists of a main memory, an associative memory, and a paging disk memory. It is organized as a virtual memory with fixed-sized pages. There are 206 words in a page and 64 bits in a word. The main memory and the paging disk memory are divided into pages. There are 32 pages in the main memory. The associative memory has 32 words, with one word representing one page in the main memory. By means of the associative memory, the virtual address is translated into the physical address.

The pages are linked into page lists. There are many types such as available page list, user's page list, space available list and in-core list. A page is brought into

TABLE I—Symbol Memory Operations

Symbol	Memory Operation
KP	no operation
AG	assign group
IG	insert group
FF	fetch and follow
FR	fetch and reverse follow
FL	follow and fetch
FD	fetch direct
SA	store and assign
SO	store only
SI	store and insert
SD	store direct
DE	delete to end
DS	delete string
DL	delete list
RG	reclaim group

the main memory upon demand and returned to the same page location on disk when it is purged. Paging is managed by the hardware system supervisor.

Hardware linked structure *

Each page in the main memory has three regions: the page header, the group-linked words, and the data space group. The page header occupies 4 64-bit memory words. Each group link word occupies one word. Each data space group has eight consecutive memory words (which is the smallest unit for data memory allocation).

There are 28 group linked words and 28 data space groups with one group link word for each data space group. The link word stores the forward and backward links of the data space group. By means of these link words, the data space groups are doubly linked into list structures. The page header stores the list heads.

There is a Memory Controller associated with the main memory. This Memory Controller can perform 14 memory operations which are shown in Table I. These operations are available to all of the processors in the system, and are used in memory word linking and memory space allocation, but the processors do not have to directly keep track of the memory addresses.

Data types and structures

The memory structure can recognize numbers and strings. The numbers are of variable-length, packed-decimal, floating-point numbers, and carry a precision designation. Each string is represented by the special character SS, followed by a variable number of characters and terminated by the special character SE. Note that the information related to the data is carried and stored with the data itself. Source programs are a special form of strings.

The memory structure allows a linked data structure, which can be a varying length group of data elements where the data element can be a number, a string, or an address link to another group of data elements. With this recursive definition, this structure can represent a vector, a matrix, or a linked structure. An example of the matrix description by symbol language and representation by this structure is shown in Figure 11. Note that the special characters in Figure 11 are listed in Table II.

As another example, a source string and its object string of an assignment start name table are shown in Figure 12. The storage of the source string, the object string, and the name table, are shown in Figure 13. The object string is composed of name table addresses, literal data, operators, and links to the source string. Each identifier in the name table is associated with a control word. All references in the object string of the identifier point to the corresponding control word. The object string and the name table are totally independent of the future size and the data type of the variable.

MEMORY STRUCTURE FOR HIGH-LEVEL LANGUAGE MACHINES

High-level language programmers need to describe the high-level data structures such as stack, queues, tables and files as often used in the software design. If a highly descriptive, high-level programming language is used to write the program, this programming language should have language constructs that are capable of explicitly describing these high-level data structures. The hardware memory structure of a computer offers the structure for storing the data elements and the data structure that are specified in a program. If this is a high-level language memory structure, it should be capable of directly imaging these high-level language constructs.[16]

A memory structure is conceived for high-level language machines.[15] This structure is shown in Figure 14. It consists of a virtual memory, an associative memory and a data interpreter. The virtual memory stores the data elements, while the associative memory stores the structure names, the data types, the struc-

TABLE II—Special Symbols in SYMBOL System

special characters	name
SS	string start char
SE	string end char
EV	end vector
CW	control word
<	left group mark
>	right group mark
|	field mark

* Figures 11 through 13 are taken from Reference 12.

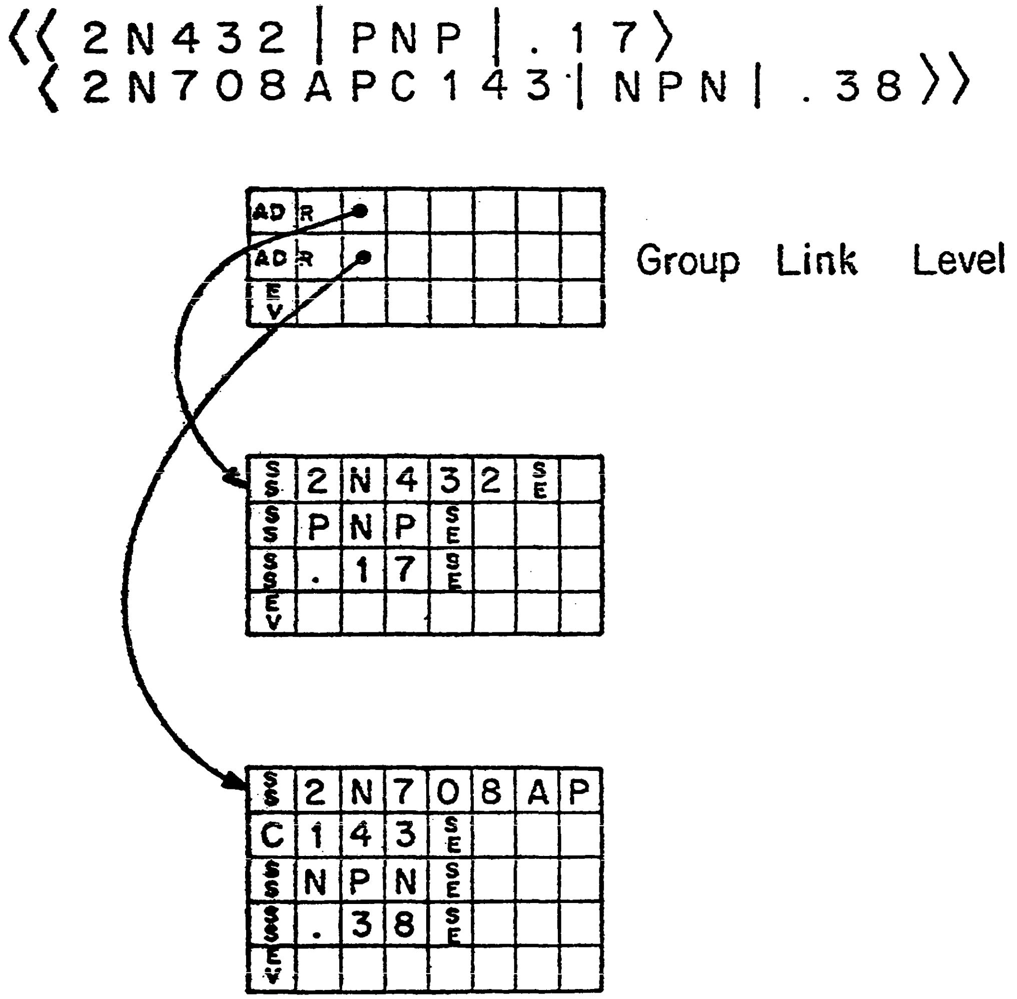


Figure 11—Symbol representation of a simple two dimensional array and its storage as three variable length memory strings

ture types and other necessary information. The data interpreter makes entries into the associative memory when the high-level language declarations are scanned. It then interprets the data when the data names are referenced. This is now illustrated for the data structures of stacks and table.

Source String:

Alpha ←— Beta * 3.2 − (Long Name join Beta);

Object String

A[Alpha] A[Beta] 3.2 * A[Long Name] A[Beta] join − ←— ;

Figure 12—Source string and object string of a symbol assignment statement

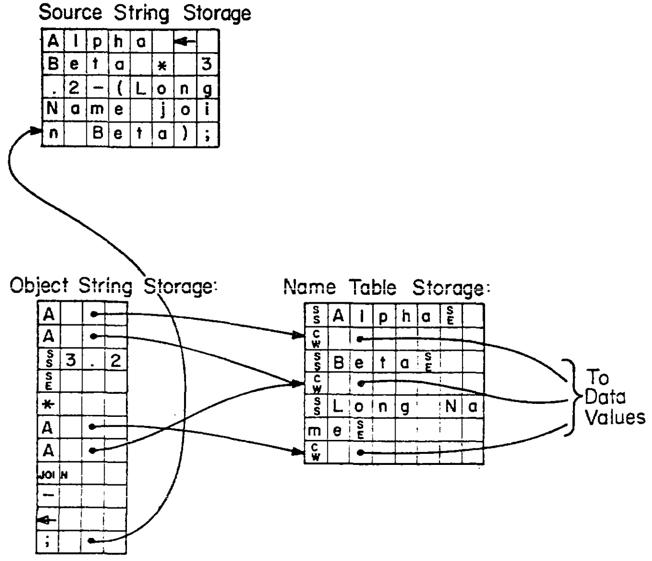


Figure 13—Storage of a source string, its object string and its name table

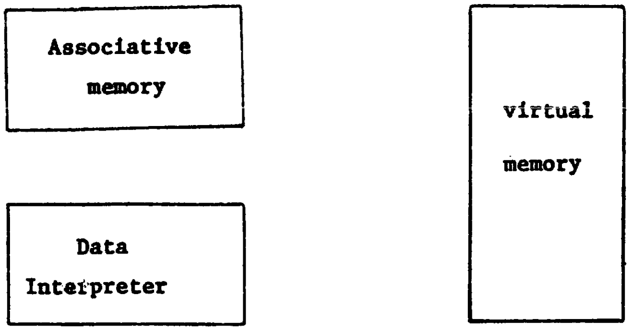


Figure 14—A hardware organization for data interpretation

Stacks

Stack is a vector with a changing number of data elements. Only at one end of the stack can the element be added (or pushed down) or deleted (or popped up), and only one element can be added or deleted at one time. A pointer is normally associated with the stack; it points to the top element where addition or deletion is permitted. It is a first-in-last-out structure. Each data element of the stack may have one or more fields; in this respect, a stack appears like a table. A stack is highly useful for handling subroutine calls, in evaluating expressions, and in traversing a tree.

Stack declaration

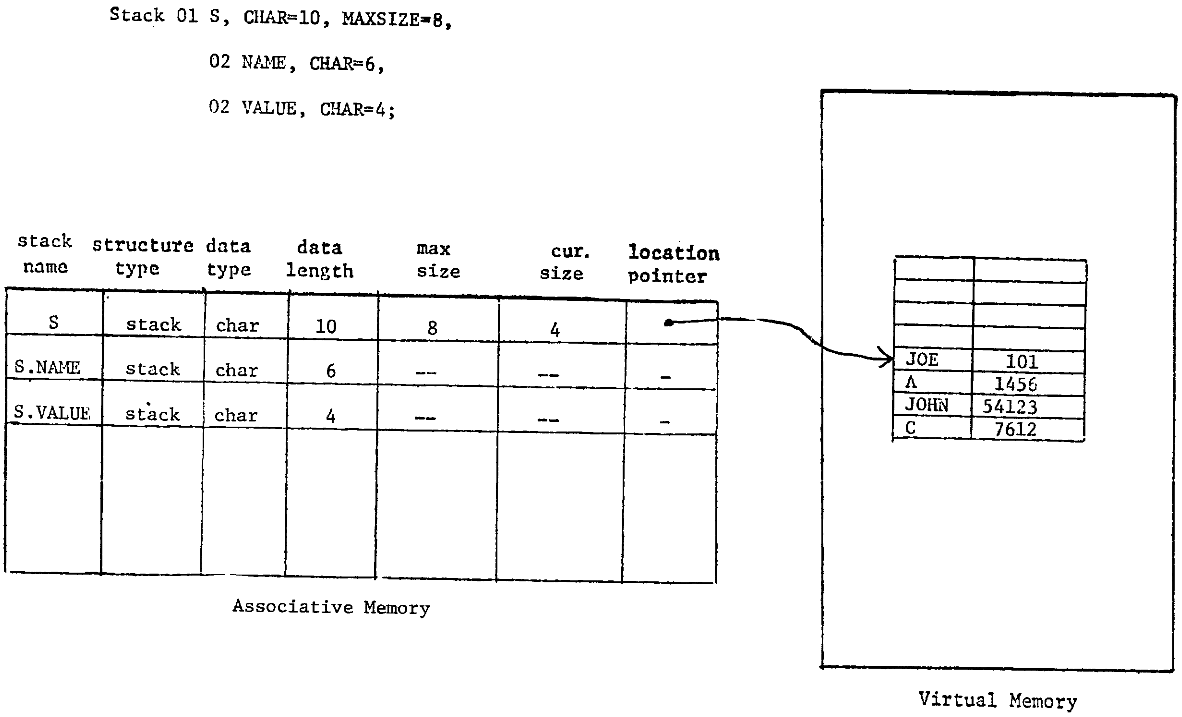
The present high-level programming languages have no explicit declarations for declaring the structure of a stack. In the case of Fortran, for example, a one-dimensional array is declared to describe the stack. Such a deficiency in the language construct is due to the inhibition of the language designer toward the conventional memory structure, though the stack should be and could be declared as descriptive and as concise as possible. Shown below is an example of a stack declaration:

```
Stack 01 S, CHAR=10, MAXSIZE=8,
            02 NAME, CHAR=6
            02 VALUE, CHAR=4
```

This statement declares a stack whose name is S, whose data element is 10 characters, and whose maximum number of elements is 8. Each stack element has two fields, the NAME field of 6 characters and the VALUE field of 4 characters.

Stack internal representation

The structural information of the above stack S can be stored in the associative memory as the three entries shown in Figure 15. Each entry has 7 fields. The first five fields store the stack name, the structure type,



```
Stack 01 S, CHAR=10, MAXSIZE=8,
        02 NAME, CHAR=6,
        02 VALUE, CHAR=4;
```

stack name	structure type	data type	data length	max size	cur. size	location pointer
S	stack	char	10	8	4	•
S.NAME	stack	char	6	—	—	-
S.VALUE	stack	char	4	—	—	-

JOE	101
A	1456
JOHN	54123
C	7612

Figure 15—Internal representation of stacks

the data type, the data length and the maximum length of the stack. The current size field stores the relative location of the top element of the stack. The pointer field stores the memory address of the top element of the stack.

Interpretation of a stack declaration is to generate associative memory entries as shown in the flow chart of Figure 16. As shown, when a declaration is recognized, it recognizes the first token of the declaration. This token reveals the structure type. It can be a stack, a table, a queue, etc. In case it is a stack, it recognizes the stack id, continues to scan the declaration, creates one or more entries for the stack, and stores the entries in the associative memory. A memory allocator is called to allocate the main memory space for the stack. This process is repeated until no more stack id is found in the declaration.

Notice the closeness between the stack declaration and the stack internal representation; this closeness means a simple interpretation of the stack declaration.

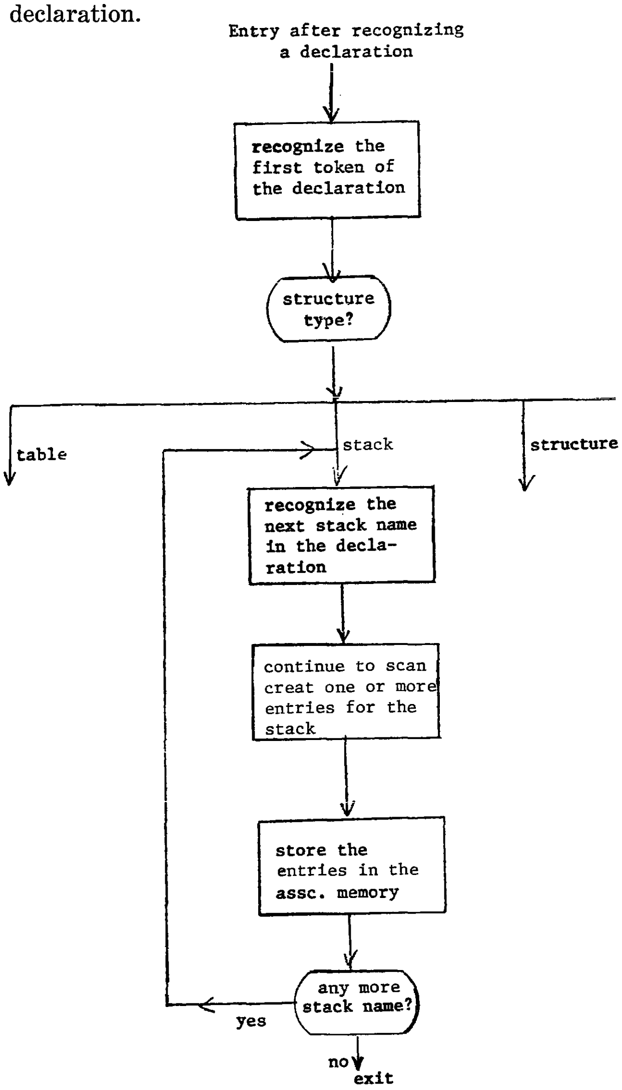

Entry after recognizing a declaration
recognize the first token of the declaration
structure type?
table
stack
structure
recognize the next stack name in the declaration
continue to scan creat one or more entries for the stack
store the entries in the assc. memory
any more stack name?
yes
no
exit

Figure 16—Interpretation of a stack declaration

Stack references and operations

After the stack declaration is interpreted, the declared stacks can now be referenced. Some examples of referencing the previously-declared stack S are shown below.

```
B:=S(TOP);
B:=pop S;
B:=S(TOP)+S(TOP−1);
A:=S(TOP,NAME);
```

where B and A are char-string buffers. The first statement assigns the value in the top element of stack S to buffer B; the value in the top element is not changed. The second statement pops out the top element of stack S to buffer B; the original value in the top element is lost. The third statement stores the sum of the values of the two topmost elements in buffer B. The fourth statement assigns the value in the NAME field of the top element to buffer A.

Interpretation of stack reference and stack operation during the direct execution of a high-level language program is shown in the flow chart of Figure 17. As shown, when an id is recognized, the associative memory is searched for the id. A message is printed

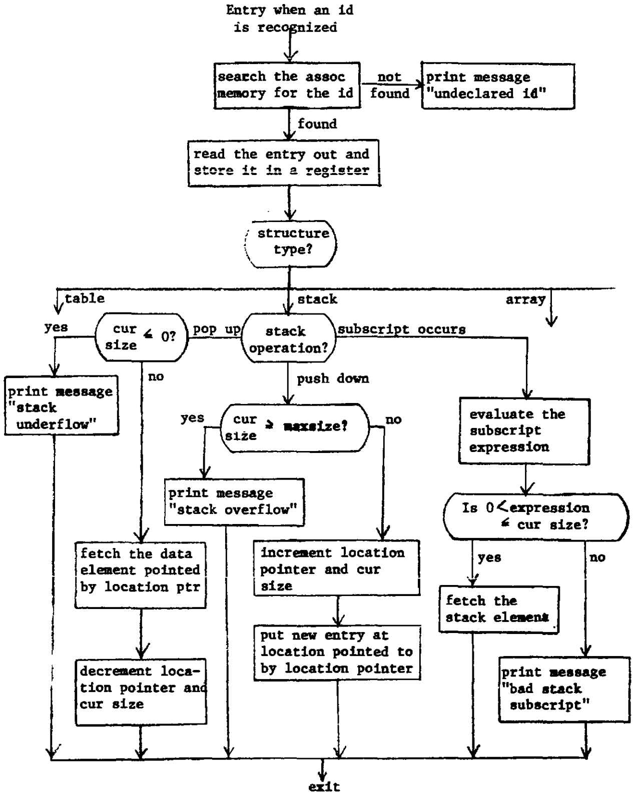

Entry when an id is recognized
search the assoc memory for the id
not found
print message "undeclared id"
found
read the entry out and store it in a register
structure type?
table
stack
array
yes
cur size ≤ 0?
pop up
stack operation?
subscript occurs
no
push down
print message "stack underflow"
yes
cur size ≥ maxsize?
no
evaluate the subscript expression
print message "stack overflow"
Is 0<expression ≤ cur size?
fetch the data element pointed by location ptr
increment location pointer and cur size
yes
no
fetch the stack element
put new entry at location pointed to by location pointer
decrement location pointer and cur size
print message "bad stack subscript"
exit

Figure 17—Interpretation of a stack operation or a stack reference during the direct-execution of a high-level language program

out if the id is not found. When it is found, the entry is read out and stored in a register. When the structure type is recognized from the register to be stack, the interpreter scans for the required operation.

(a) If it is a stack pop-up operation, the stack underflow is checked. If there is none, the data element pointed by the location pointer in the register is fetched from the virtual memory. Both the current size and the location pointer are then decremented.
(b) If it is a push-down operation, the stack overflow is checked. If there is none, both the location pointer and the current size are incremented and the data element is then stored in the virtual memory.
(c) If subscript TOP is found, it is a reference operation; the subscript expression is evaluated. If the value of the expression is within the current size, the stack element is fetched from the virtual memory; otherwise, a message is printed to indicate a bad stack subscript.

Tables

Tables are two dimensional arrays which permit different data types among the fields (or columns) of the table. A table can be of a fixed or varying size. All entries of the table are accessible for examinations or modifications. The entry of a table can be located by using a subscript or by matching an argument with the contents of a field of the table.

Table declaration

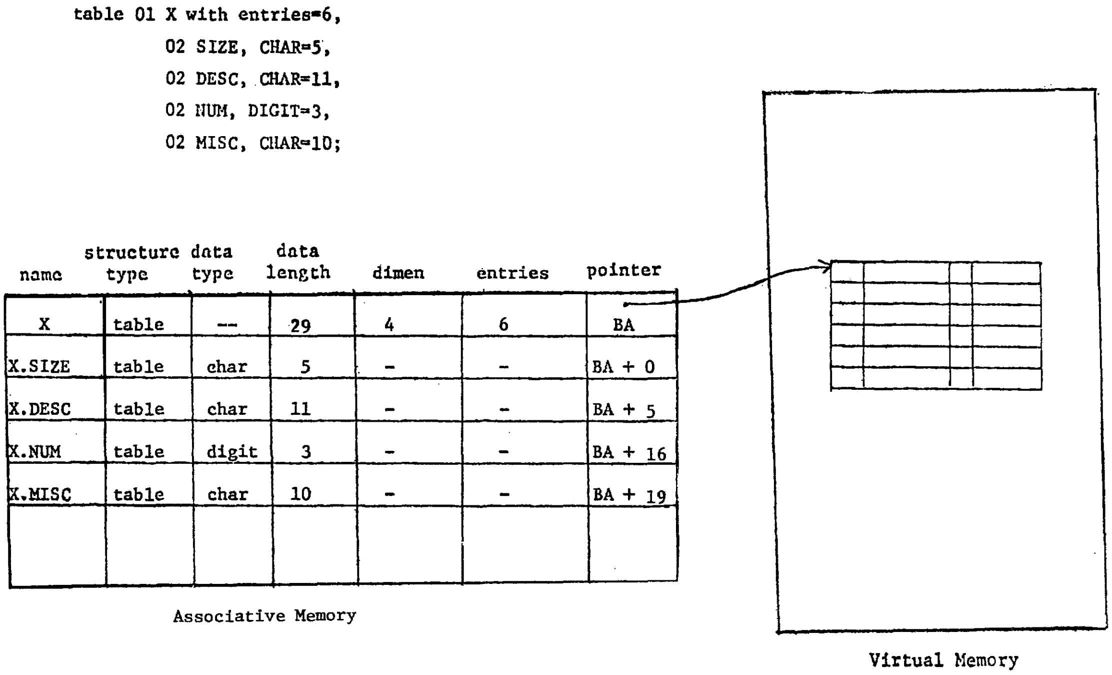
The current high-level languages have a structure declaration which can declare the table structure. However, a table cannot be declared explicitly and directly. An example of the table declaration is:

```
Table 01 X with entries=6,
         02 SIZE, CHAR=5,
         02 DESC, CHAR=11,
         02 NUM, DIGIT=3,
         02 MISC, CHAR=10;
```

This statement declares a table with name X which has 6 entries. There are four fields in each entry: the SIZE field of 5 characters, the DESC field of 11 characters, the NUM field of 3 digits and the MISC field of 10 characters. For simplicity, one digit and one character are assumed of the same length.

Table internal representation

The structural information of table X can be stored in the associative memory as the five entries shown in Figure 18. Each entry has 7 fields. The first four fields



```
table 01 X with entries=6,
         02 SIZE, CHAR=5,
         02 DESC, CHAR=11,
         02 NUM, DIGIT=3,
         02 MISC, CHAR=10;
```

name	structure type	data type	data length	dimen	entries	pointer
X	table	—	29	4	6	BA
X.SIZE	table	char	5	-	-	BA + 0
X.DESC	table	char	11	-	-	BA + 5
X.NUM	table	digit	3	-	-	BA + 16
X.MISC	table	char	10	-	-	BA + 19

Figure 18—Internal representation of a table

store the table name or the column name, the structure name, the data type and the data length. The dimen field and the entries field specify the numbers of columns and rows in the table, respectively. The pointer field stores the location of the table in the main memory. Note that the value of the data length field is the sum of the data lengths of all the fields of the table. Again, the syntax and the semantics of the table declaration are very close to the internal representation, allowing a simple interpretation of the table declaration.

Table references

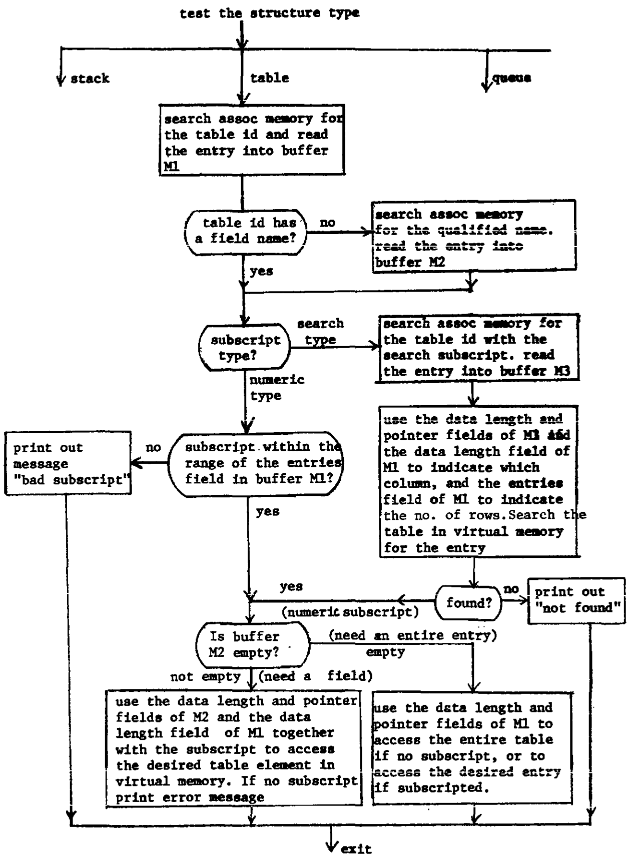
After the table declaration is interpreted, the declared table may now be referenced. Some examples of table referencing are:

```
X[2] := ("4x3", "TILE", "BLACK") ;
X.SIZE[3] :"JOHN";
B:=X[X.DESC='JOHN'];
B:=X.SIZE[X.DESC='JOHN'];
```

The first statement stores strings "4x3", "TILE" and "BLACK" to the SIZE, DESC, and MISC field of the second row of table X, respectively. (Nothing is changed in the NUM field.) The second statement stores string "JOHN" to the SIZE field of the third row of the table. The third statement searches for that entry in table X whose DESC field contains string "JOHN" and assigns this entry to buffer B. The fourth statement performs the same operation as the third statement except that the fourth statement assigns the SIZE field of the found entry to buffer B. The above four examples show that the table can be referenced either by a numerical subscript (in a square bracket) or by a search subscript (a qualified name indicated by a dot between the table name and the field name).

Interpretation of a table reference during the direct execution of a high-level language program is shown in the flow chart of Figure 19. After the identifier is recognized to be a table name, the associative memory is searched for the table name. (e.g., X); if it is found, this entry is read out of the associative memory and placed in buffer M1. Next, it is determined whether the table name is a qualified name (i.e., name with a field name). If it is, the associative memory is again searched and the entry is now read into buffer M2. Finally, the subscript type is determined.

(a) If it is a numeric subscript, it is checked to see whether the subscript is within the range.
(b) If it is a search subscript, the associative memory is searched for the search subscript, and the found entry is read into buffer M3. Then, the data length and pointer fields in buffer M3 and the data length field of buffer M1 are used to determine which column is located the table element. The entries field of buffer M1 is used to indicate the number of rows. The table in virtual memory is then searched for the entry. If this is not found, a "not found" message is printed out. Otherwise, the subscript is converted into a numeric subscript.



test the structure type
stack
table
queue
search assoc memory for the table id and read the entry into buffer M1
table id has a field name?
no
search assoc memory for the qualified name. read the entry into buffer M2
yes
subscript type?
search type
search assoc memory for the table id with the search subscript. read the entry into buffer M3
numeric type
use the data length and pointer fields of M3 and the data length field of M1 to indicate which column, and the entries field of M1 to indicate the no. of rows. Search the table in virtual memory for the entry
print out message "bad subscript"
no
subscript within the range of the entries field in buffer M1?
yes
yes
(numeric subscript)
found?
no
print out "not found"
Is buffer M2 empty?
(need an entire entry)
empty
not empty
(need a field)
use the data length and pointer fields of M2 and the data length field of M1 together with the subscript to access the desired table element in virtual memory. If no subscript print error message
use the data length and pointer fields of M1 to access the entire table if no subscript, or to access the desired entry if subscripted.
exit

Figure 19—Interpretation of a table reference during the direct-execution of a high-level language program

At this time, either of the above two cases has a numeric subscript. If buffer M2 is empty, it calls for a field of the table entry; otherwise, it calls for an entire entry.

(a) If a field is needed, the data length field and the pointer field of buffer M2 and the data length field of buffer M1 together with the subscript are used to access the desired table element in the virtual memory. If there is no subscript, an error message is printed out.
(b) If an entire entry is needed, the data length field and the pointer field of buffer M1 is used to access the entire table if there is no subscript, or to access the desired entry if there is a subscript.

At this point, the reference of a table name in either of the above cases is completed.

Other data structures

The memory structure shown in Figure 14 has been described for the declarations, the internal representations and the references of stack and table. This memory structure can similarly be used for other high-level data structures such as buffers, arrays, queues, structures and files.[15]

CONCLUDING REMARKS

Since the advent of commercial electronic digital computers, the memory has undergone an evolution during the past 20 years. It has made great hardware advances in speed, capacity, size, cost and reliability. The increase in memory speed has been realized not only by electronical technology but also in memory organization by using a memory buffer (i.e., cache memory). The increase in memory capacity has been realized not only by electronical technology but also in memory organization by using a one-level memory (i.e., virtual memory). The memory has also made significant structural changes in recognizing data elements and data structures. It can recognize number (fixed and/or floating-point binary and decimal), character strings for a set of 64 characters, and logical words. However, the hardware implementation of data structure is practically none except the memory structures of two recently available, commercial computers, where a hardware stack (this stack is not for the programmer to use in his program) and hardware arrays are implemented.

Where do we go from here? The memory structure of a computer has a great influence on the computing and processing capability of the computer. The usage of this capability is through programming. The programmer makes use of the memory structure to satisfy his software needs in implementing such data elements as numbers, characters, strings, pointers, and program segments and such data structures as stacks, tables, queues, arrays, tree, buffers, structures, and files. Therefore, it is apparent that the future memory structure would advance toward satisfying such software needs, and the hardware implementation of these data elements and data structures represents one of the most challenging structural advances in the forthcoming computer structures.

REFERENCES

1. *IBM System/360 Principles of Operation,* IBM System Reference Library, File No. S360-01, GA22-6821-7, 1970.
2. Burroughs Corporation, "A Narrative Description of the Burroughs B5500 Disk File Master Control Program," Detroit, Michigan, October, 1966.
3. Burroughs Corporation, "Burroughs B5500 Information Processing Systems Reference Manual," #1021326, Detroit, Michigan, May 1969.
4. Burroughs Corporation, "B1700 Systems Reference Manual," #1057155, Detroit, Michigan, April 1972.
5. Burroughs Corporation, "Burroughs B6500 Information Processing Systems Reference Manual," #1043676, Detroit, Michigan, September 1969.
6. Burroughs Corporation, "Burroughs B6700 Handbook, Volume 1," #5000276, Detroit, Michigan, January 1972.
7. Burroughs Corporation, "Burroughs B7700 Information Processing Systems Reference Manual," #1060233, Detroit, Michigan, January 1973.
8. Digital Equipment Corporation, "PDP 11/45 Processor Handbook," 1971.
9. Intel Corporation, "MCS-8 8008 8-Bit Parallel Central Processor Unit, Users Manual," Santa Clara, California, November 1973.
10. Intel Corporation, "Intel 8080 Microcomputer System Manual," Santa Clara, California, 1975.
11. MacKenzie, F. B., "Automated Secondary Storage Management," *Datamation,* November 1965, pp. 24-28.
12. Smith, William R., et. al., "SYMBOL—A Large Experimental System Exploring Major Hardware Replacement of Software," *Proceedings of the Spring Joint Computer Conference,* pp. 601-616, 1971.
13. Liptay, J. S., "Structural Aspects of the System/360 Model 85 II The Cache," *IBM Systems Journal,* Volume Seven, Number 1, 1968, pp. 15-21.
14. Organick, Elliott I., *Computer System Organization,* Academic Press, Inc. 1973.
15. Yaohan, Chu and E. R. Cannon, *High-level Language Memory Structure,* Technical Report TR-409, Department of Computer Science, University of Maryland, September 1975.
16. Chu, Y., (Editor), *High-level Language Computer Architecture,* Academic Press, Inc. 1975.

Cache system design in the tightly coupled multiprocessor system

by C. K. TANG
IBM Corporation
Endicott, New York

ABSTRACT

Cache is a fast buffer memory between the processor and the main memory and has been extensively used in the larger computer systems. The principle of operation and the various designs of the cache in the uniprocessor system are well documented.[1-9] The memory system of multiprocessors has also received much attention[10-17] recently; however, they are limited to the systems without a cache. Little if any information exists in the literature[18-20] addressing the principle and design considerations of the cache system in the tightly coupled multiprocessor environment. This paper describes such a cache design. System requirements in the multiprocessor environment as well as the cost-performance trade-offs of the cache system design are given in detail. The possibility of sharing the cache system hardware with other multiprocessing facilities (such as dynamic address translation, storage protection, locks, serialization, and the system clocks) is also discussed.

CACHE SYSTEM REQUIREMENT OF MULTIPROCESSOR SYSTEM*

In a multiprocessor system, the main store is shared by all the processors. A single copy of the operating system in the shared main store controls the entire system. Since the main store can be accessed by all the processors, it is also used as the 'mailbox' to pass messages between the processors.[10,16] In addition to the shared main store, some multiprocessor systems may use the 'private stores', each of which can be accessed by only one processor.[10,17] 'Private stores' are used to store tables of frequently used subroutines, tables for allocation of private resources, etc. Since the 'private stores' do not communicate with each other, they are not to be confused with the caches in the multiprocessor system. Furthermore, a multiprocessor system with cache (such as the IBM System/370 Model 158 MP and Model 168 MP) will not need the 'private stores', because the cache will provide the performance advantage that the 'private stores' offer. As a result, this paper will not address the 'private stores'.

A multiprocess system can have multiple tasks dispatched simultaneously, and each processor can independently execute its instructions. Hence from each processor's standpoint, it will have the complete access to the main store without the awareness of the existence of the other processor(s).[16] The logical relation between the processors and the main store can be shown in Figure 1. The I/O channel is an I/O processor; thus, from the main stores point of view, the I/O channel is logically equivalent to the other instruction processors. Figure 1 can also represent the 'timeshare' or 'common bus' main store system design.[17] Although 'crossbar switch' and 'multiple-bus' are frequently used in the main store system design for performance reasons (Figure 2 and Figure 3), the logical relation between the processors and the main memory remains the same. If a cache is attached to each processor, then logically the caches should be considered as a part of the main store, since each cache should be transparent to each processor in the same way that the cache is transparent to the processor in a uniprocessor system. Figure 4 describes such a relationship. The channel usually doesn't need a cache for performance reasons, but will be involved in the cache system design to maintain the cache transparency to the channel. Comparing Figure 4 to Figure 1, one can easily see that the communications among the caches and the channel are necessary so that the main store system will still preserve the logic view of Figure 1 to the processors. The basic cache design itself (e.g., the directory, the set associativity, the block sizes, etc.) should be the same as the cache of a uniprocessor.[1-8] In the following section, communication among the caches and the channel in the multiprocessor system will be given in detail, as well as the portion of the basic cache design that relates to such communication.

CACHE SYSTEM DESIGN IN THE MULTIPROCESSOR SYSTEM

In a typical uniprocessor cache design,[5] a directory is used to translate the main store addresses to the

* NOTE: Hereafter, all references to the "multiprocessor system" will mean "tightly coupled multiprocessor system."

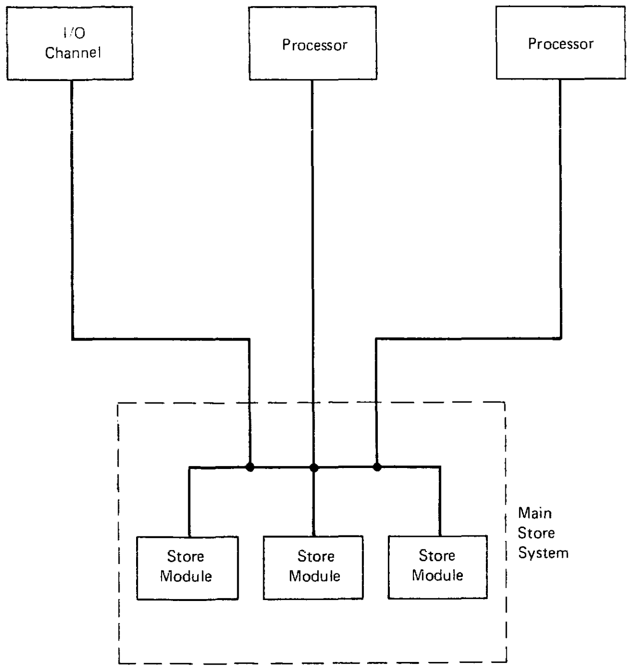


Figure 1—The logical relation between processors and the main store

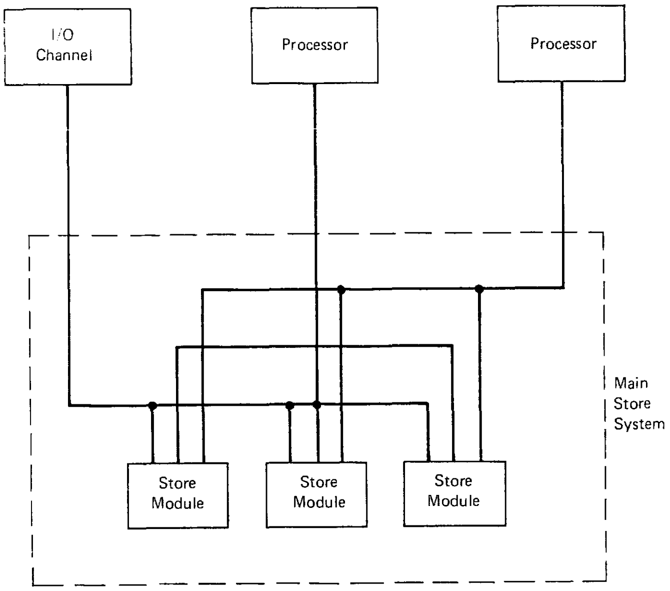


Figure 3—The multiple-bus main store system

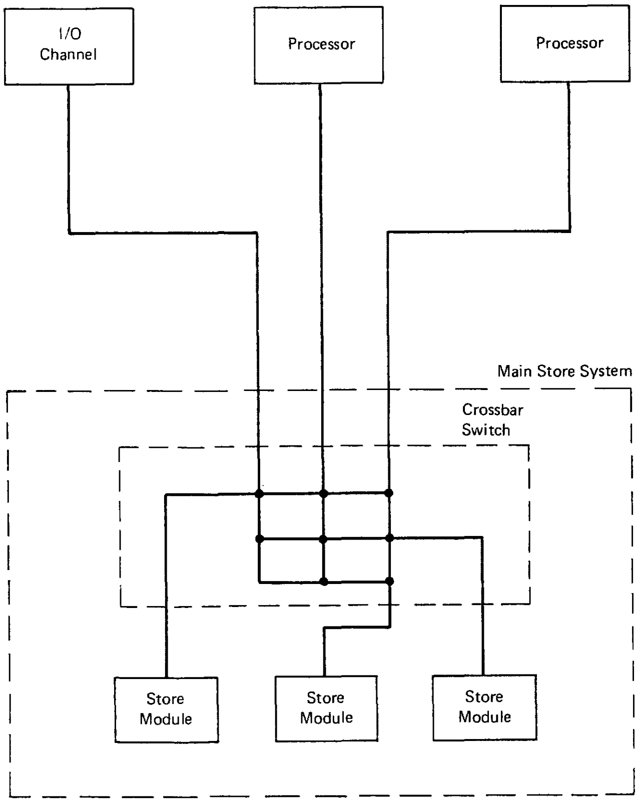


Figure 2—The crossbar main store system

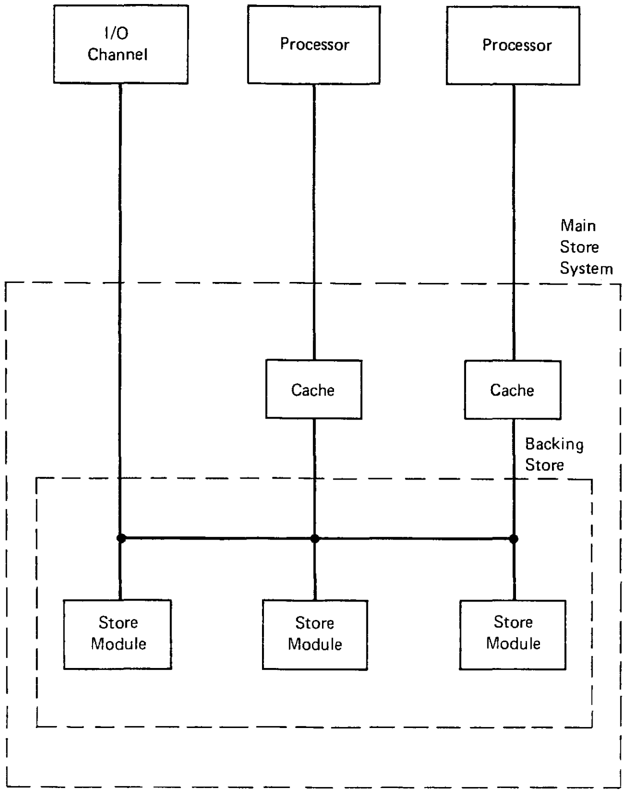


Figure 4—A multiprocessor system with caches

cache location of the data block in cache. Figure 5 shows the logical organization of such translation. No hardware detail is given since various designs are possible[1] and is irrelevant to the description of the subject issues.

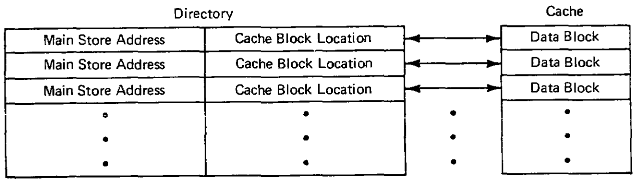
In a multiprocessor system, the same cache organizations can be used for each processor, with the additional controls to facilitate the communication among the caches and the channel. The following definitions are needed to describe the communication:

Block the unit of translation, which is usually also the unit of data transfer between the cache and the main store.

Line block(s) that are associated with a given main store address. In a uniprocessor cache, a line is a block, whereas in the multiprocessor system, a line can exist in more than one cache. For example, a main store address refers to 64 bytes of data, and these 64 bytes of data exist at two caches, which can be addressed by each correspondent processor using the line address.

Shared Line a line that exists at the cache(s), which has not been modified (written) with respect to the copy at the backing store by the processor. The line is transferred to the cache from the backing store when the processor makes *read* reference and misses the line in the cache. A shared line should be allowed to exist simultaneously in more than one cache so that 'read only' data can be accessed more efficiently in the multiprocessor system.

Private Line a line that exists in a cache which has been modified (with respect to backing store) or is going to be modified by its corresponding processor. A private line should exist in only one cache so that at any moment, throughout the system, only one version of data exists for any address. This is a requirement implied by Figure 4 (cache transparency).



Directory			Cache
Main Store Address	Cache Block Location	◄──►	Data Block
Main Store Address	Cache Block Location	◄──►	Data Block
Main Store Address	Cache Block Location	◄──►	Data Block
•	•	•	•
•	•	•	•
•	•	•	•

Figure 5—Cache organization in a uniprocessor

Status bits would have to be used in each entry of the cache directory to identify each line (shared or private).

There are many 'store algorithms' in a cache design.[7] In this section, the 'store only in cache' algorithm is used. It means that if the processor wants to write and a miss occurs in cache, then the line is always brought to the cache so that the processor can always write to cache. Minor changes would be necessary to the description below if another algorithm is used.

To control the communication among the caches and the channel, a 'store controller' is necessary (Figure 6). A set of commands from the caches to the store controller and another set of commands from the store controller to the caches are defined below. To perform the control functions, the store controller would have to know, at all times, the status of every cache, namely, what lines (shared or private) exist at which cache. This can be done in two ways: (1) use the store controller 'central directory' to keep track of every line in each cache, or (2) interrupt the caches to find out. The former approach certainly has the performance advantage, especially when the system has more than two processors. The description of the commands given below corresponds to the former approach; obviously a change of description should be made if the latter approach is used.

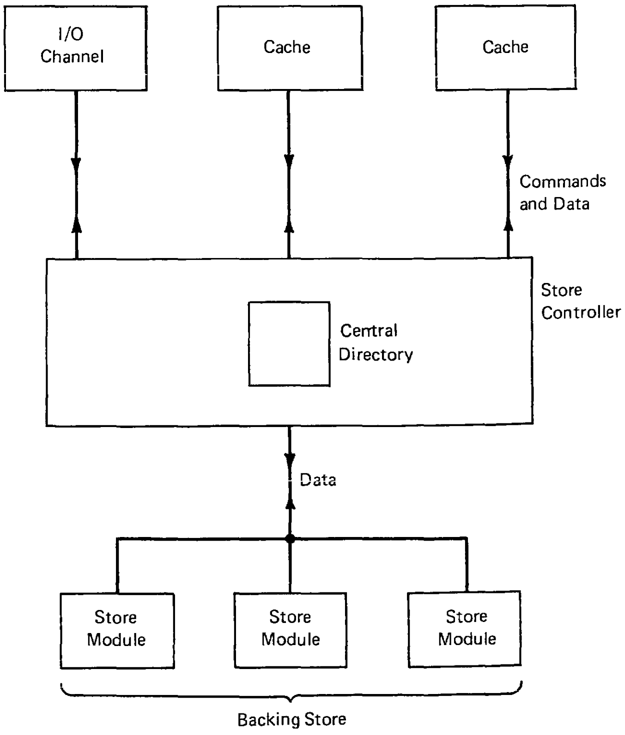


Figure 6—Store controller and control directory

COMMAND FROM CACHE TO THE STORE CONTROLLER

- Shared Read—This command is used when the processor wants to read (the main store) and a cache fault (miss in the cache) occurs. This command will signal the store controller to bring the line from the backing store to the cache. The line will be marked as a shared line in the cache directory and the central directory should record the fact that this cache (the cache that issues this command) contains the line as a shared line.
- Private Read—This command is used when the processor wants to write and a cache fault occurs. This command will signal the store controller to bring the line from the backing store to the cache. The line will be marked as a private line in the cache directory. The store controller should make sure (by using the store controller-to-cache commands described later) that only this cache has this line as a private line; no other caches can have this line shared or private. Such a fact should also be recorded in the central directory.
- Declare Private—When the processor wants to write, and if the line exists in its cache as a private line, the processor can write to cache and the line will remain private. No communication to the store controller would be necessary in such a case. On the other hand, if the line exists in its cache as a shared line, the line has to be converted to a private line before the writing to the cache can take place. Thus, this cache will change the status of the line in its directory from shared to private, and will inform the store controller of the change using this command. Upon receiving the command, the store controller will record the change in the central directory. In addition, the store controller will examine the central directory. If the central directory indicates that some other cache(s) has (have) this line as a shared line, the store controller will use the store controller-to-cache command (described later) to remove such a line from the cache(s). The removal is necessary since a private line in a cache means no other cache can have the same line shared or private. No data transfer is involved in this command.
- Replace Private Line—Replacement of the line in a cache is necessary to make room in the cache for the new line coming from the backing store. If the line being replaced is a private line, the backing store has to be updated using this command since the data in the cache have been modified with respect to the backing store. In addition to the data transfer from the cache to the backing store, the removal of the private line from this cache should also be reflected in both the cache directory and the central directory.
- Replace Share Line—If the line being replaced is a shared line instead of a private line, no data transfer from the cache to the backing store is necessary. The cache directory and the central directory should reflect such removal, and this command is used to signal the store controller to update its central directory for the removal. In hardware design, this command can be issued simultaneously with either the 'shared read' or the 'private read' for performance reasons, since these two commands cause the replacement.

COMMANDS FROM THE STORE CONTROLLER TO THE CACHE

- Unprivate Line—When some other cache issues a Share Read command to the store controller, the store controller will search its central directory for any cache that contains the line involved as a private line. If such a cache is found, this command will be issued to the cache found. Upon receiving this command, the cache directory of cache found will be changed to indicate that the line exists now as a shared line. The store controller will also update its central directory to reflect such a change. In addition, the line will be transferred from this cache to the backing store for updating since a private line contains modified data with respect to the backing store. This command is also used when the channel reads the main store directly.
- Remove Private Line—When some other cache issues a Private Read command to the store controller, the store controller will search its central directory for any cache that contains the line involved as a private line. If such a cache is found, this command will be issued to the cache. Upon receiving this command, the cache will remove the line from its directory. The store controller will also update its central directory to reflect such a change. In addition, the line will also be transferred from this cache to the backing store for updating. This command is also used when the channel writes to the main store directly.
- Remove Shared Line—When some other cache issues a Private Read command to the store controller, the store controller will also search its central directory for any cache(s) that contains the line involved as a shared line. More than one cache may contain such a line. The store controller will issue this command simultaneously to the cache(s) that contains such a line. Upon receiving this command, the cache(s) will remove the line from its (their) directory (directories). The store controller will also update its central directory to reflect such change(s). No data transfer from the cache to the backing store is necessary. This command is also used when the channel writes to the main store directly.

It should be clear now that by using the set of eight commands outlined above, in conjunction with the basic uniprocessor cache design, the multiple-cache system can meet the cache transparency requirement described previously. The channel still addresses the store controller using the ordinary 'Read Main Store' and 'Write Main Store' commands, which will invoke the store controller to issue the store controller-to-cache commands to maintain the cache transparency. For example, when the channel issues a Read Main Store command to the store controller, the store controller has to search its central directory for any caches that contain the line involved as a private line. If found, the Unprivate Line command is issued to the cache that contains it, and the store controller will also update its central directory to reflect such change. The backing store also has to be updated by the private line in the cache before the data can be transferred to the channel. Similar operation by the store controller is necessary when the channel issues the Write Main Store command.

CONCLUSION

A practical cache system design for the multiprocessor system is outlined in this paper. Although it does not represent exactly any of the cache system designs of the existing multiprocessors, it does illustrate the system requirement and design concept of the cache system. The use of a store controller was chosen (for managing the central directory, etc.) instead of using the interrupt approach. The store controller approach certainly requires more hardware, but offers much better performance especially when the multiprocessor system contains more than two processors. This should be apparent from the fact that a simple Share Read issued by a cache will cause the interrupt of all the other caches in the interrupt approach.

Many other system elements have to be handled with extra hardware facilities in a multiprocessor environment in addition to the hardware in a uniprocessor. These include the dynamic address translation, storage protection, locks, serialization, system clock, etc.[16,17] They all require communication between the caches/processors. This extra hardware can be conveniently incorporated into the store controller and can be used for these functions. These shared uses of the store controller would make the cost of the store controller less formidable.

ACKNOWLEDGMENT

The author wishes to thank many of his colleagues for the numerous discussions on the subject issues.

REFERENCES

1. Conti, C. J., "Concepts for Buffer Storage," *IEEE Computer Group News,* Vol. 2, No. 8, March 1969, pp. 9-13.
2. Liptay, J. S., "Structure Aspects of the System 360 Model 85, II-The Cache," pp. 15-29, *IBM System Journal,* 7/1/68.
3. *A Guide to the IBM System/370 Model 165,* IBM Corporation Form GC20-1730, 1970.
4. *A Guide to the IBM System/370 Model 155,* IBM Corporation Form GC20-1729, 1970.
5. Katzan, H., Jr., "Storage Hierarchy System," *Proceedings of the Spring Joint Computer Conference,* 1971, pp. 325-336.
6. Kaplan, K. R. and R. O. Winder, "Cache-Based Computer Systems," *Computer,* Vol. 6, No. 3, March 1973, pp. 30-36.
7. Bell, J., D. Casasent and C. G. Bell, "An Investigation of Alternative Cache Organizations," *IEEE Transaction on Computers,* Vol. C-23, No. 4, April 1974, pp. 346-351.
8. Meade, R. M., "On Memory System Design," *Proceedings of the Fall Joint Computer Conference,* 1970, pp. 33-42.
9. Laliotis, T. A., "Main Memory Technology," *Computer,* Vol. 6, No. 8, August 1973, pp. 21-28.
10. Searle, B. C. and D. E. Freberg, "Microprocessor Applications in Multiple Processor Systems," *Computer,* Vol. 8, No. 10, October 1975, pp. 22-30.
11. Juliussen, J. E. and F. J. Mowle, "Multiple Microprocessors with Common Main and Control Memories," *IEEE Transactions on Computers,* Vol. C-22, No. 11, November 1973, pp. 999-1007.
12. Kurtzberg, J. M., "On the Memory Conflict Problem in Multiprocessor Systems," *IEEE Transactions on Computers,* Vol. C-23, No. 3, March 1974, pp. 286-293.
13. Bhandarkar, D. P., "Analysis of Memory Interference in Multiprocessors," *IEEE Transactions on Computers,* Vol. C-24, No. 9, September 1975.
14. Sastry, K. V. and R. Y. Kain, "On the Performance of Certain Multiprocessor Computer Organizations," *IEEE Transactions on Computers,* Vol. C-24, No. 11, November 1975, pp. 1066-1074.
15. Baer, J. L., "A Survey of Some Theoretical Aspects of Multiprocessing," *ACM Computing Surveys,* Vol. 5, No. 1, March 1973, pp. 31-80.
16. Mackinnon, R. A., "Advanced Function Extended with Tightly-Coupled Multiprocessing," *IBM System Journal,* Vol. 13, No. 1, 1974, pp. 32-59.
17. Enslow, P. H., Jr., *Multiprocessors and Parallel Processing,* John Wiley & Sons, New York, 1974.
18. *A Guide to the IBM System/370 Model 158,* IBM Corporation, Form GC20-1754.
19. *A Guide to the IBM System/370 Model 168,* IBM Corporation, Form GC20-1755.
20. Noguchi, K., I. Ohnishi and H. Morita, "Design Consideration for a Heterogeneous Tightly Coupled Multiprocessor System," *Proceedings of the National Computer Conference,* Vol. 44, May 1975, pp. 551-559.

Coupling small computers for performance enhancement*

by FERNANDO C. COLON, ROBERT M. GLORIOSO, WALTER H. KOHLER and DOMINIC W. LI
University of Massachusetts
Amherst, Massachusetts

ABSTRACT

The advent of the microprocessor has opened up new avenues for the system designer to provide more powerful, more reliable and more user oriented computer systems to the user community for the same or lower costs. The problem confronting the designer is: How to achieve these goals? This paper describes one such method called Distributed Function Multiple Processor (DFMP).

The system described uses several micro processors each with its own memory to form a cluster. These processors are differentiated by the functions they perform such as file managing, intelligent terminal, etc. and communicate via a Restricted Cross Bar Switch (RCBS). Further, several clusters or nodes can be linked to form a local network. Interprocessor and internode communications are controlled by a special processor called the Interprocessor Controller (PC) located in each node. The IPC's use an adaptive technique to determine traffic flows.

INTRODUCTION

Rapid advancements in IC technology have increased the performance of logic circuits in terms of speed, gate density, reliability, and power consumption. Also, quantity production of LSI (Large Scale Integration) chips has made complex microprocessors and large memories available at low and decreasing costs. Such developments are beginning to force changes in some basic concepts of computer system design. Already, there exists computers on-a-card which include a reasonably fast CPU and basic memory. This cost is only a relatively small part of the total system cost. In many instances, peripheral devices and their controllers cost more than the CPU and basic memory. Therefore, the traditional concept of keeping the CPU and central memory fully utilized is becoming less and less important. One trend in computer design is to replace multitask monoprocessor systems with multiprocessor systems. These multiprocessor systems improve cost-performance by sharing expensive peripheral units. Computer networking is another approach that has become popular. A set of remote computers are interconnected through communication links in order to share resources. The potential advantages of building both type of systems include: (1) the sharing of resources (data, programs, special hardware); (2) reliability and availabiiity; and (3) performance enhancement.

Although multiprocessors and computer networks can be conceptually described as a set of interconnected nodes, they are significantly different in terms of the amount of "coupling" present.[1] Multiprocessor systems are "tightly" coupled in the sense that the processors share memory, I/O devices, and execute procedures under a single integrated operating system. Data in multiprocessors can be effectively transferred by the movement of pointers, as all processors share the same physical address space. Enslow[2] provides a historical perspective on the development of multiprocessors and discusses most of the major multiprocessor systems being considered or implemented today, and Baer[3] provides an excellent survey of the most important aspects of multiprocessing.

Computer networks, as exemplified by several loop systems and the ARPA Network,[4-6] are "loosely" coupled. There is no sharing of memory and the interconnecting media are fixed communication links where a datum of information is sent serially at rates of 100 to 50 Kbaud between nearly autonomous computers. While each computer can operate independently, it has higher effective capability when networked to the others. The papers by Farber[7] and Pyke[8] review the highlights of computer network technology and examine seven typical networks. Most of the research conducted in the area of computer networks that relates to this paper can be classified as follows: (1) models and analytic methods for network design;[9] (2) routing strategies;[10-12] and (3) coordination of processes and protocols.[13-15]

The multi-user, multi-task, *d*istributed *f*unction, *m*ultiple-*p*rocessor system DFMP to be described is

* This work was supported in part by a grant from the Digital Equipment Corporation.

designed to take advantage of the emerging technologies and to incorporate features of both multiprocessors and networks in an integrated manner. The basic element of DFMP is a microprocessor with its own memory. Collections of microprocessors are connected through a restricted cross bar switch to form a node of the system. Each node operates as a multiple-processor system which is coordinated by a special processor called the INTER PROCESSOR CONTROLLER (IPC).

The use of microprocessors in multiple-processor systems is certainly not new. (See the work by Ravindran[16] and Jordan[17].) However, research in this area has concentrated on "tightly" coupled systems and the problem of mismatch between microprocessor and memory cyle times. We have taken another approach. In DFMP each microprocessor is called a functional unit and is similar to a Computer Module (CM).[18] Basically, the functional units are processor-memory pairs with several special ports (connections to the restricted cross bar switch) or bus interfaces. There is no central shared memory in the sense of C.mup or HSM Imp[10] and the physical address space is the sum of all local memories. Thus, we will see that the functional units are more tightly coupled than the nodes of a computer network, yet less tightly coupled than the processors in conventional multiprocessor systems. The resulting intermediate coupling is a distinctive characteristic of the DFMP system, and for this reason we shall refer to it as a multiple-processor system.

The *a priori* design constraints for the system were:

(1) low cost processors must be used;
(2) the cost of the control and communications must each be limited to the cost of a single processor;
(3) the operating system must be incorporated as much as possible into the firmware and hardware;
(4) the system must be expandable with minimal modification to the operating system;
(5) to each user the system must appear as a medium powered general purpose computer.

Additional details of the design are presented in the following sections which are organized as follows. The second section discusses the system architecture; the third section is concerned with the execution of user's jobs in the system and the deadlock problem, and finally, the fourth section summarizes the concepts incorporated in our design.

SYSTEM ARCHITECTURE

An initial global view of DFMP reveals a network of independent computer systems, called nodes, interconnected by explicit communication links. The network may be a geographically localized one (i.e., installed in the same room) or a geographically distributed one (the nodes are far apart, i.e., in different states). The advantages of localizing the network are lower communication cost, simple network interfacing hardware, richer connection of the nodes, and a simpler communication routing procedure. In this initial phase of the design, a localized network is assumed, although the design principles involved apply to a geographically distributed network as well.

Figure 1 shows the block diagram of DFMP. For illustrative purposes only three nodes are shown, although in practice the number of nodes is only limited by the cost of the interconnecting cables and communications hardware. Each node is a distributed function, multiple-processor computer system consisting of three basic components: (1) a collection of Functional units; (2) an Interprocessor Controller (IPC); and (3) a Communications Switch. The main functions and the hardware structure of each element are described below.

(1) *Functional Units*—The functional units are the basic building blocks of a node. As will be described in the next section, jobs submitted by the users are dynamically partitioned into subtasks which are executed by specialized functional units. The hardware structure of a typical functional unit is depicted in Figure 2. It consists of a processor, local memory and interprocessor communication hardware. Each unit belongs to a particular class depending on the functions it performs, and may have additional hardware or

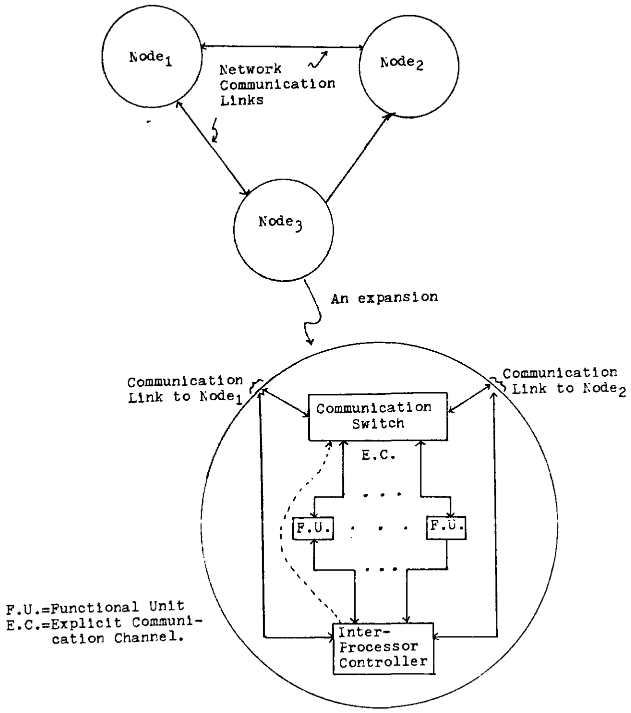

Node$_1$
Network Communication Links
Node$_2$
Node$_3$
An expansion
Communication Link to Node$_1$
Communication Link to Node$_2$
Communication Switch
E.C.
F.U.
F.U.
Inter-Processor Controller
F.U.=Functional Unit
E.C.=Explicit Communication Channel.

Figure 1—A global view of DFMP reveals a network of computer systems, called nodes. Each node is itself a multiple-processor system

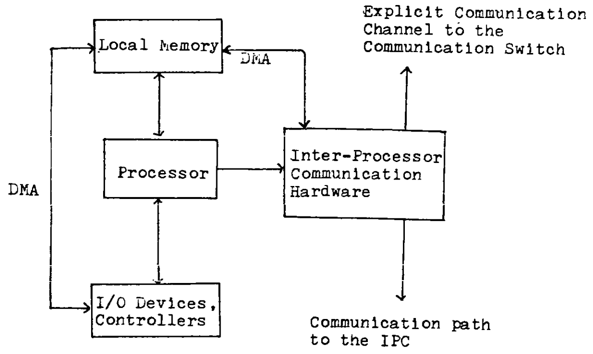

Explicit Communication Channel to the Communication Switch
Local Memory
DMA
Processor
Inter-Processor Communication Hardware
DMA
I/O Devices, Controllers
Communication path to the IPC

Figure 2—Block diagram of a typical functional unit. While all functional units have the processor, local memory, and interprocessor communication hardware, the I/O devices and controllers are present in some functional units only

software peculiar to that class. For example, if the functional unit is specialized in Input/Output functions, it will also have its own I/O devices and I/O controllers as well.

(2) *Interprocessor Controller (IPC)*—The functional units can be regarded as autonomous system resources which are allocated and deallocated throughout the process of running a user's job. The main purpose of the IPC is to serve as the resource manager and network coordinator for a node. The hardware of the IPC (Figure 3) consists of a relatively fast processor and a local memory which stores the IPC control functions and the resource allocation tables. All local functional units are connected to the IPC via a direct Communication Bus. If the node is networked, the IPC also has (i) control lines for the Network Communication Links and (ii) a direct communication path to adjacent IPC's for network coordination.

(3) *Communication Switch*—The function of the Communication Switch is to provide for the interprocessor communications (both intra-node and internode) among the functional units. Several alternatives for the switch were examined: a fast processor serving as a communication processor; a common bus; and different kinds of cross-bar switches. Considering their efficiency in terms of speed, cost, reliability, availability and ease of networking, a Restricted Cross-Bar Switch (RCBS) was chosen. The cross-bar switch is restricted in the sense that

(a) only one-to-one connections are allowed, but there can still be many different one-to-one simultaneous connections;

(b) data transfer is done bit serial, synchronously through the switch, but the bit serial rate is high and the transfer time of one memory word is of the same order as the processor instruction cycle time.

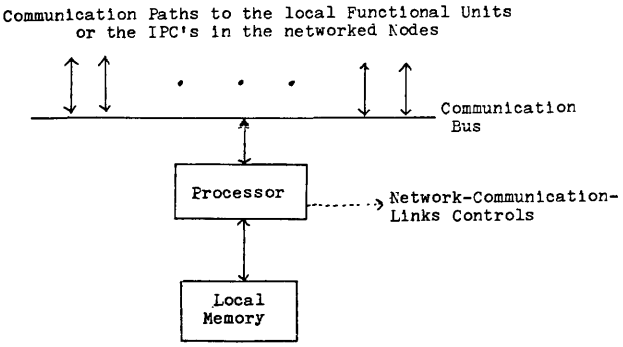

Communication Paths to the local Functional Units or the IPC's in the networked Nodes
Communication Bus
Processor
Network-Communication-Links Controls
Local Memory

Figure 3—Block diagram of the IPC

The RCBS consists of the following four parts (see Figure 4(a):

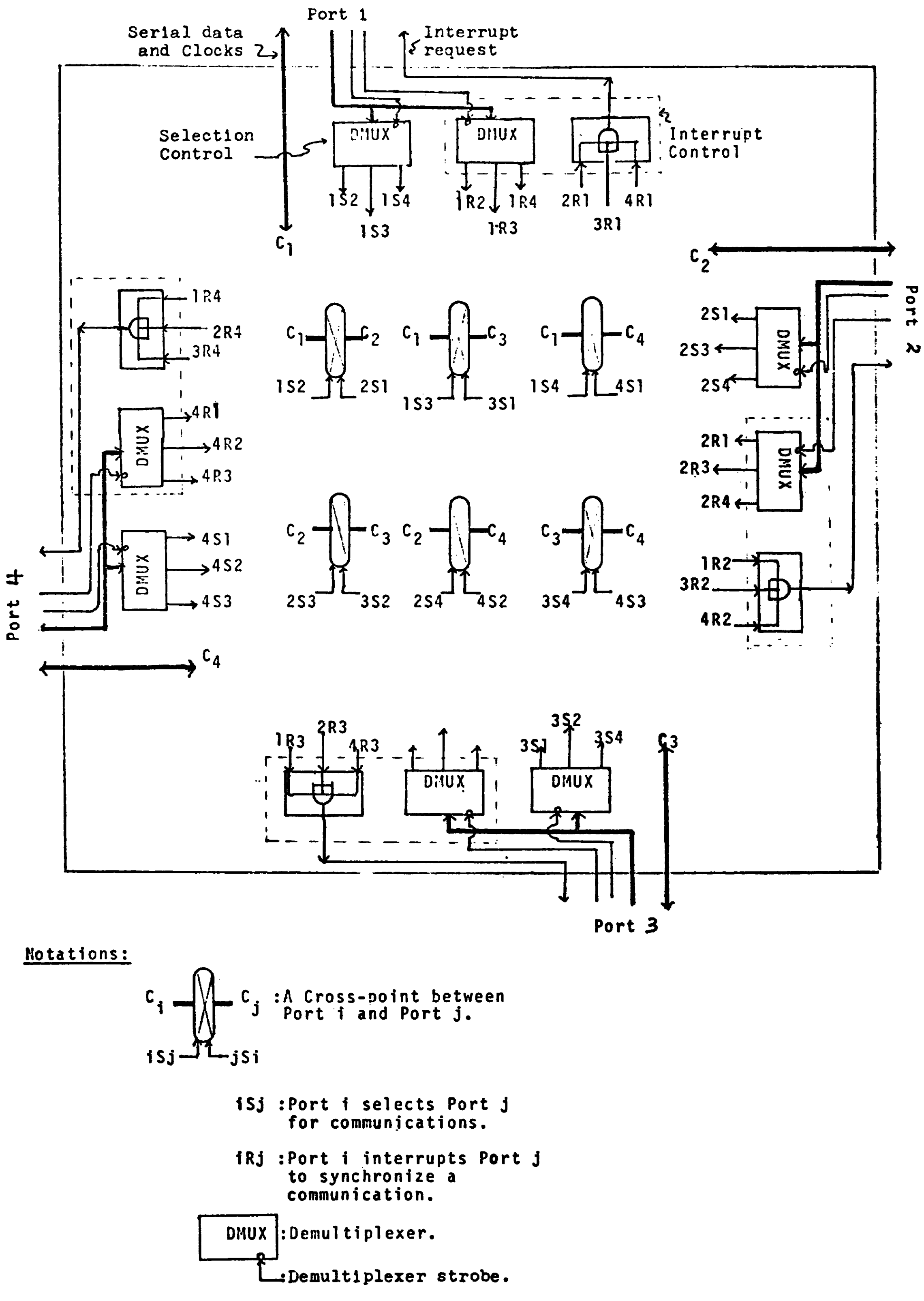
(a) *Port*—A port may be connected to a local functional unit, or a port of another node to form a network link. Physically, a port is simply a socket in the RCBS module.

(b) *Cross-points*—A cross-point can be considered as a unit cell in the RCBS. The number of cells in a RCBS having N ports is $N(N-1)\div 2$. They are interconnected in such a way that each port can be connected to any other port by "turning on" a single cross-point. After turning on the particular cross-point, signals can be transmitted between the pair of ports. As shown in Figure 4(b), a cross-point can be simply made from four open-collector NAND gates and a single NOR gate. A full-duplex bit serial transmission path is accomplished by using two NAND gates to pass the bit serial data plus the clocking signal in each direction. The remaining NOR gate is used to turn on the cross-point when the input conditions are met (See below).

(c) *Port Selection Logic*—The port selection logic consists of 1-of-N demultiplexers, with one for each port. The input signals from a port to the demultiplexer contain the name of the port to which a connection is desired. The outputs of the demultiplexers are connected to the respective cross-points in such a way that a cross-point is turned on if and only if each port selects the other.

(d) *Interrupt Logic*—The interrupt logic also consists of 1-of-N demultiplexers, one for each port. The input to a demultiplexer gives the name of the functional unit to be interrupted. The use of this interrupt logic enables the functional units to synchronize their communications without going through the IPC.

The example in Figure 5 shows how the RCBS is used to facilitate both inter-node and intra-node interprocessor communication. In the next section we will describe how the different components of DFMP interact to carry out a user's jobs.



Notations:

:A Cross-point between Port i and Port j.

iSj :Port i selects Port j for communications.

iRj :Port i interrupts Port j to synchronize a communication.

:Demultiplexer.

:Demultiplexer strobe.

Figure 4(a)—A RCBS having four ports

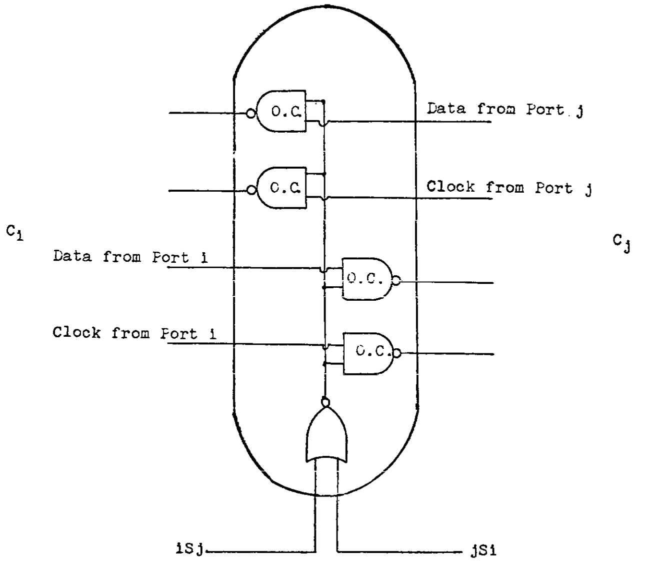


Figure 4(b)—A cross-point between port i and port j. The controls iSj and jSi are normally high. The cross-point is turned on if and only if both lines are low (when both ports select each other)

JOB PARTITIONING

The processor-memory pairs described earlier are grouped into different classes according to the functions they perform. Units from each class are requested as needed and coordinate their actions to carry out the execution of jobs submitted by users. This is an example of the division of labor method introduced in the GAMMA 60 design and further explored by Foster[20] and Spier.[21] Although the number of different classes and the mix of units is application dependent, in a general purpose environment there are five readily identifiable classes:

(1) *Intelligent Terminal* (*I.T.*)—An intelligent terminal is assigned to each user and remains dedicated to that user until his requests are completed and he relinquishes the unit. An I.T. enables interaction with the system and provides support to the user by performing functions such as command prompting, line editing, message buffering and command interpreting.

(2) *General I/O*—A functional unit in this class handles I/O devices such as line printers, paper tape readers/punchers, and cassettes. It performs I/O transfers between its devices and other functional units. This unit alleviates the problem of mismatched speeds between functional units and I/O devices by using a spooling system to handle all I/O requests.

(3) *Execution*—A unit in this class responds to the requests of an I.T. or another execution unit to execute a user-generated computational procedure. The microprocessors in this group may have extra hardware such as an extended arithmetic element. Also, they have special routines for linking and loading the appropriate segments of a user's program.

(4) *Program Development*—A unit in this class executes sharable re-entrant system programs such as compilers, assemblers, and text editors that are stored in local ROM's. It receives requests from other functional units and performs software support functions.

(5) *File Management*—A unit in this class manages a secondary storage system (disk, drum, extended core, etc.) and executes file functions such as "CREATE", "DELETE", "RENAME", "APPEND", and so forth. Furthermore, a File Manager stores and retrieves state information on logical tasks which have been suspended. The details of the "suspension" mechanism will be given in the next section.

Next, let us consider the actions and coordination of functional units in the execution of a user's task.

Resource allocation

In order to understand the coordination primitives used by the system and the different levels of interaction present among the functional units, we will follow a user's task through the system. First, the user logs into the system and is assigned an Intelligent Terminal. This I.T. will serve as the user's representative in his transactions with the system. The I.T. is the only functional unit whose binding to the user and user's job is permanent and thus independent of such factors as system loading, failure of other system functions, and so forth.

In general, the I.T. receives commands for the execution of jobs through a *job control language*. Typical commands would be: RUN, COMPILE, CREATE FILE, and EDIT. Also, each command would include an aggregate of information to make the command executable. For example, the "COMPILE" command would include the name of the compiler needed, the name of the source code, and the I/O devices involved. The I.T. interprets this system command language and transforms it into a sequence of subtasks and resource requests. In this way the "RUN" command would be transformed into the following sequence of subtasks:

(1) Compile (SOURCE NAME/LOC, OBJECT NAME/DEST)
(2) Execute (OBJECT NAME/LOC).

These subtasks may, during their execution, create other subtasks, i.e., the Compile task above would probably generate subtasks that involve file transfers and I/O listings. This description completes the discussion of the first level of interaction present in the system, that is, user to functional unit interaction.

The subtasks created by the system commands are considered as a series of resource requests. Hence, it

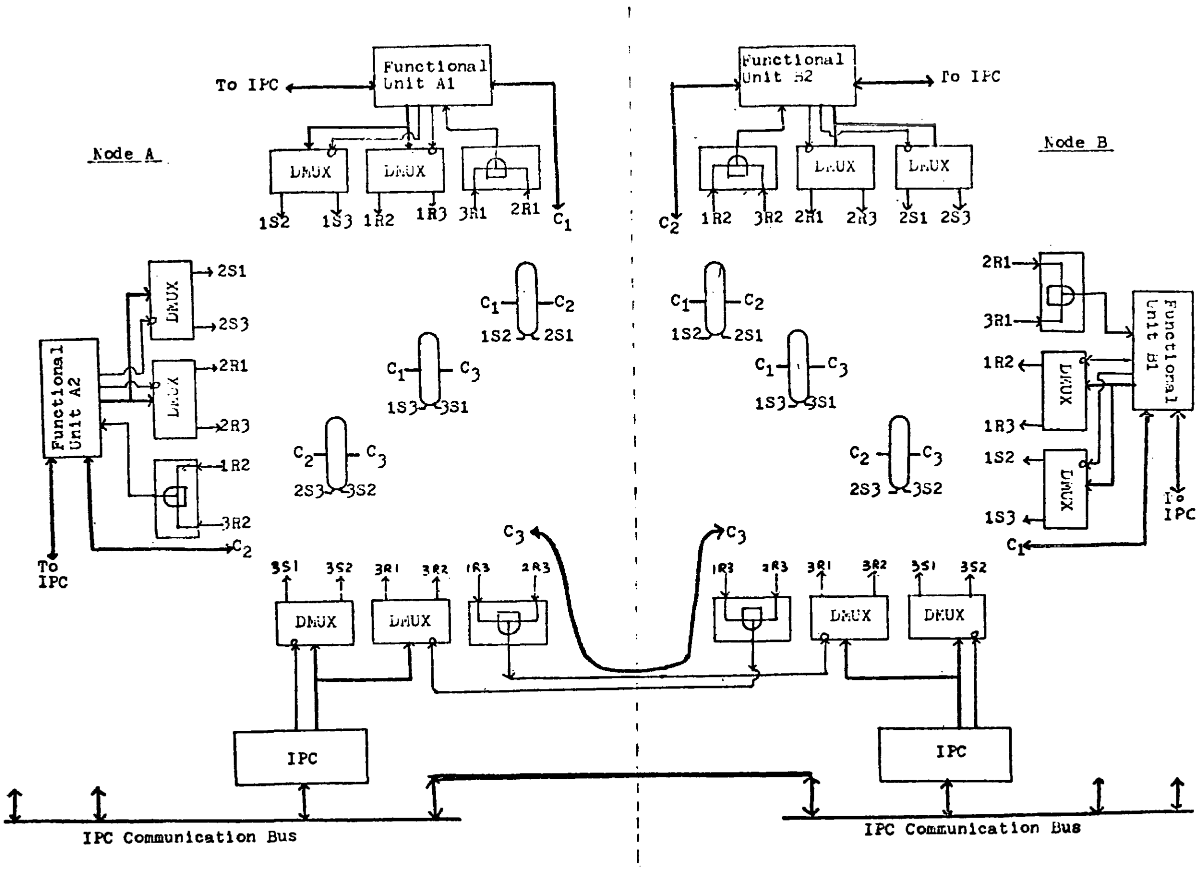


Figure 5—An example on the use of the RCBS to form a two nodes network. If a port is connected to a functional unit, the inputs to the demultiplexers (for the selection and interrupt controls) are controlled directly by the functional unit. If a port is used to form a network link, the demultiplexers are controlled by the IPC's in such a way that the network link is made transparent to the functional units in the two nodes

is the responsibility of the I.T. to initiate these requests by posting them with the resource allocator (the IPC) within the node. This interaction between functional units and IPC represents the second level of interaction. Such interactions are needed whenever tasks executing within the functional units come to a point where additional resources are needed to complete their work. In the present example the "COMPILE" task would require the allocation of a Program Development unit (P.D.) in order to complete its work. Such requests must be posted with the IPC.

Upon acknowledgment of the request, the IPC would initiate a search for a free unit of the resource type requested. This search involves a two-stage decision, where first a node and then a specific functional unit within that node is selected. The selection of the node will be based on the adaptive scheme of Glorioso and Colon.[22] The IPC makes its node selection using a set of network statistics. These network statistics are a measure of the likelihood of obtaining the desired resource for each node (including the local node) present in the system. Following the selection of a node, the requesting IPC will inform the selected node's IPC of its request. This involves a third level of interaction, namely IPC to IPC.

Within the selected node, a much simpler method is used to select a specific functional unit to service the request. Each node has an associated Availability Status Register for each of its resource classes. This register contains the status information (either busy or free) of each functional unit in the particular resource class. Therefore, the decision at this stage is made by a scan of the corresponding register for a free unit followed by allocation of this unit, if one is found. If no such unit is currently available, the pending request will be queued with all other requests in that node waiting for the particular resource type. This information and the queue length will be passed back to the node that originated the request (not necessarily a different node) to be used in the updating process of its network statistics. For a more detailed description of this process, see References 22 and 23.

The advantages provided by the selection scheme are as follows:

(1) The need for constant exchange of complete status information between nodes is eliminated. Considering the number of functional units within a node and a richly connected network, constant status information exchange can seriously degrade the performance of the system.

(2) Automatic load leveling is accomplished. That is, when functional units of a certain class are used up, the likelihood of selecting the corresponding node is "dynamically adjusted" by the adaptive scheme. Hence, the load will be diverted to other nodes in the system.

(3) Failsoft behavior is realized. Since failure of nodes will be reflected in the likelihood of service statistics from that node, the IPC's will avoid failed nodes.

Once a functional unit is selected, the selected unit is informed (by the IPC in the selected node) of the identity of the requesting unit, and the requesting unit is informed (by the IPC in the requesting node) of the identity of the selected unit. At this point it is the responsibility of the two units to establish a communication path through the RCBS and to coordinate their actions. This process involves the fourth and last level of interaction, namely functional unit to functional unit. In our example the I.T., after receiving the identity of a P.D. unit from the IPC, will initiate the compilation by transferring the necessary control information (SOURCE NAME/LOC, OBJECT NAME/DEST) to the P.D. unit. The P.D. on the other hand will proceed with the compilation task until completion or to a point where additional resources are needed. If additional resources are needed, the sequence of steps described above will be repeated with P.D. as the requesting unit.

Notice that as the user's job proceeds through the system, it is segmented into finer and finer procedures some of which may execute in parallel. This parallelism can be graphically represented by a timing diagram and a static tree representation. When a timing diagram is used, (See Figure 6(a)) resource allocations and deallocations are shown as vertical lines and the usage of resources is depicted as horizontal lines. It reflects the dynamic history of requests and allocations. On the other hand, a static tree representation, (See Figure 6(b)) can be considered as a snapshot of the system's outstanding requests at a particular time. It shows the logical relationship between units, i.e., who initiated whom.

When a task is completed by a functional unit, the functional unit is deallocated. Allocation and deallocation of resources continues until all tasks and subtasks are finished and the user is informed. Now, let us shift our attention to the tasks and their corresponding state transition diagrams.

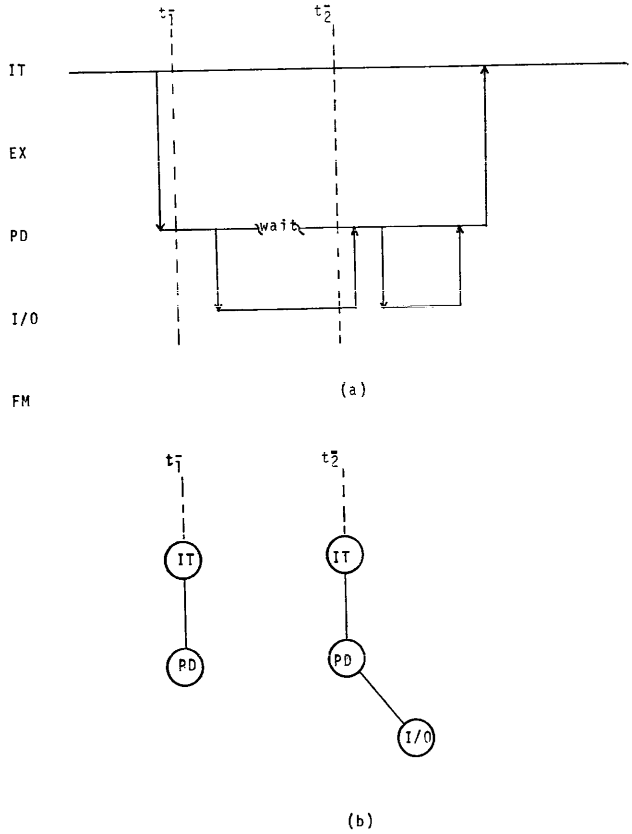


Figure 6—(a) Timing diagram representation
(b) Tree representation

STATES AND STATE TRANSITION DIAGRAMS

Tasks occupy one of the five states shown in Figure 7. A task is

(1) *terminated* before it has been created and after it has finished;

(2) *ready* if it has been created or reactivated by its parent but it is not yet bound to a functional unit;

(3) *waiting* while it is

(a) requesting additional resources for a subtask it is creating, or

(b) waiting for coordination with a subtask it has created;

(4) *running* while it is bound to a functional unit but not in the waiting state;

(5) *dormant* if it has been preempted while waiting, or after using a resource for a prespecified time limit.

Each task is created by another task, its parent. A newly created task enters a queue of tasks which are ready to run but are waiting for a functional unit. This queue of ready tasks is maintained by the IPC. When a functional unit becomes free, the IPC passes the

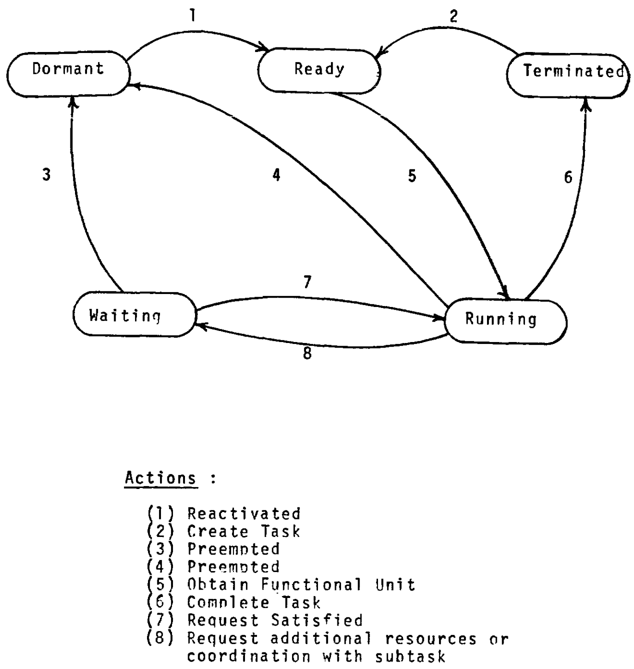


Figure 7—Task's state transition diagram

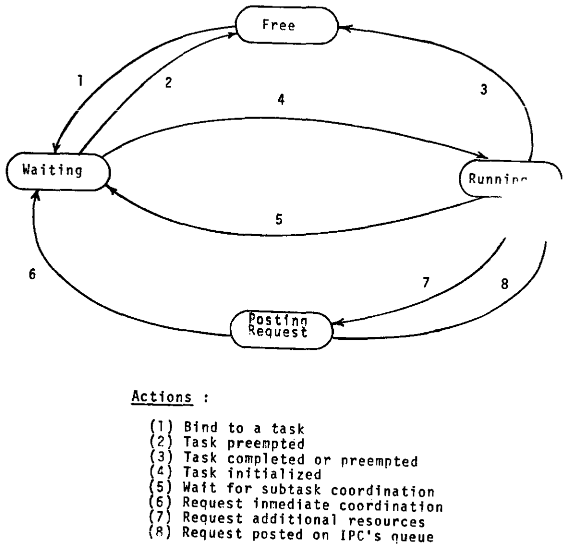

Free
1
2
3
4
Waiting
Running
5
6
7
8
Posting Request
Actions :
(1) Bind to a task
(2) Task preempted
(3) Task completed or preempted
(4) Task initialized
(5) Wait for subtask coordination
(6) Request inmediate coordination
(7) Request additional resources
(8) Request posted on IPC's queue

Figure 8—Functional unit transition diagram

unit's identity to a parent of a task waiting for that type of unit. If the free functional unit and the parent task are successful in establishing a communication path, the ready task becomes bound to the functional unit and enters the running state. Tasks in the running state may create subtasks or request coordination with tasks they have already created. To do this they must enter, and then remain in, the waiting state until the request is completed or the task is preempted (for waiting too long for example). Preempted waiting tasks enter the dormant state as do preempted running tasks. When a task enters the dormant state, its state vector is stored and its associated functional unit is deallocated. A dormant task is reactivated at a later time by its parent.

Each task is a logical entity and is only bound to a physical functional unit during the running and waiting states. To describe the status of each functional unit we will use the state transition diagram given in Figure 8. Each functional unit is in one of the following four states:

(1) *free* if it is not bound to a task;

(2) *waiting* if it is bound to a task and the task is waiting for coordination with a subtask it has created;

(3) *running* if it is bound to a task that is in the running state;

(4) *posting request* if the task to which it is bound is entering a request for additional resources.

These states parallel those of a task since each task is executed in a functional unit. The Availability Status Register bit associated with each functional unit is enabled (set to one) when it becomes free. When a unit becomes bound to a task, two hardware timers are initialized. One is enabled whenever the functional unit is in the running state and the other whenever it is in the waiting state. If either of the accumulated times exceeds the prespecified system limits, the task will be preempted and the functional unit will be deallocated once the state of its associated task has been saved to enable restart. When a running task creates a subtask, it notifies the functional unit to which it is bound to post a request for an additional resource. The functional unit will then enter the posting request state and communicate the request to its IPC. If the task is able to continue running independently of the subtask, the functional unit will return to the running state; otherwise, it will enter the waiting state. In the DFMP system, each task will interact with its parent task and its subtasks by means of a small local monitor in the functional unit to which it is bound. These monitors will be the building blocks of the distributed operating system of DFMP.

Deadlock prevention

Whenever concurrent tasks contend for system resources a condition of deadlock may occur. This phenomenon has been studied extensively in connection with the design of computer operating systems and sufficient conditions for the prevention of deadlock are well-known.[24,25] One solution to this problem is to impose a linear ordering on all resource (functional unit) classes. If a task has been allocated a functional unit in class R[i] it may subsequently create a concurrent subtask requiring a functional unit in class R[j] only if j>i. This eliminates the possibility of a circular waiting condition. The linear order we have adopted is shown in Figure 9. We can in fact relax the requirement that j>i and allow j=i if a functional unit in class R[j] is immediately available. In this case the request need not to be queued and circular wait is prevented. When a functional unit is not immediately available and j=i, the requesting task must cancel its request and proceed to execute the subtask sequentially, using the functional unit in that class which is already bound to it. This modified rule allows parallel tasks using identical resources to be created as long as the system is not saturated. If saturation occurs, the potentially parallel tasks will be forced to run sequentially.

Task preemption and reallocation of resources

Preemption of a task enables the system to avoid the monopolization of resources by a group of one or more users. Preemption occurs when a functional unit has been in the waiting or running state for more than the specified time limit. We will describe this preemption process using an example. Assume that preemption occurs at time t in Figure 6(a). A snapshot of the state of the user's job at time t reveals the concurrent task structure represented by the tree in Figure 6(b). We will assume that the preemption occurred in the Program Development unit while in the running state. Upon encountering this condition, the local monitor would initiate a series of procedures to save the state vector of the interrupted task T and to then deallocate the functional unit. These procedures would carry out the following actions:

(1) Interrupt T's parent P.

(2) P becomes the logical parent of T's subtasks $T_1,\ldots,T_n$.

(3) Interrupt $T_1,\ldots,T_n$ and notify them of change in parent.

(4) Save T's state vector.

(5) Deallocate resource.

The state of the user's job following T's preemption is shown in Figure 10.

Class	Units
R[1]	Intelligent Terminals
R[2]	Execution Units
R[3]	Program Development Units
R[4]	File Management Units
R[5]	General I/O Units

Figure 9—Hierarchy of classes of functional units

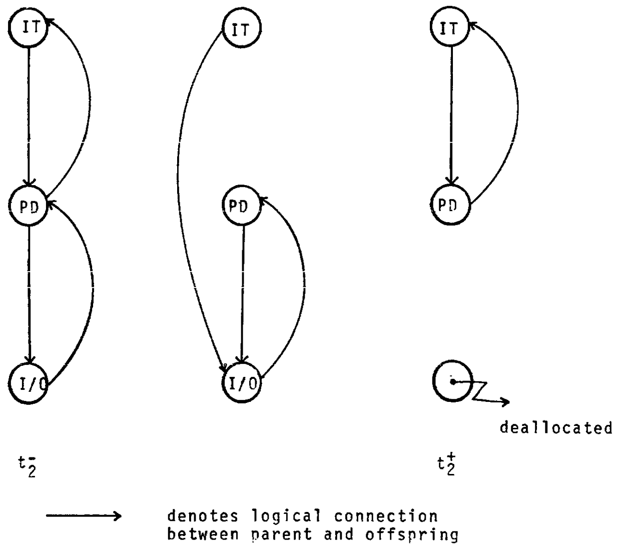


Figure 10—Task preemption

SUMMARY

The design of a networked multiple processor system, DFMP which has characteristics of both multiprocessors and computer networks has been presented. DFMP has a unique structure wherein many microprocessors each with its own memory and specialized in function are coupled to form a multiple processor. Also, several multiple processors can be joined into a computer network.

The operating system is distributed in each of the functional units. There is no central scheduler in DFMP but an Inter Processor Controller in each Multiple-Processor to coordinate communications between the functional units within and between nodes.

At this time DFMP exists only on paper. However, work on a single node using five Digital Equipment Corporation's LSI-11 Microprocessors is under way.

REFERENCES

1. Fuller, S. H. and D. P. Siewiorek, "Some Observations on Semiconductor Technology and the Architecture of Large Digital Modules," *Computer*, Vol. 6, No. 10, pp. 15-21, October 1973.
2. Enslow, Jr., P. H., *COMTRE Corporation: Multiprocessors and Parallel Processing*, John Wiley and Sons, New York, 1974.
3. Baer, J. L., "A Survey of Some Theoretical Aspects of Multiprocessing," *ACM Computing Surveys*, Vol. 5, No. 1, pp. 31-80, March 1973.
4. Pierce, J. R., "How Far Can Data Loops Go?" *IEEE Transactions on Communications*, Vol. COM-20, No. 3, pp. 527-530, June 1972.
5. Roberts, L. G. and B. D. Wessler, "Computer Network

Development to Achieve Resource Sharing," SJCC, *AFIPS Conf. Proc.*, Vol. 36, pp. 543-549, May 1970.

6. Farber, D. J. and K. C. Larson, "The System Architecture of the Distributed Computer System—The Communication System," *Symposium on Computer Network and Teletraffic*, Polytechnic Institute of Brooklyn, Vol. 22, pp. 21-27, April 1972.
7. Farber, D. J., "Networks: An Introduction," *Datamation*, Vol. 18, No. 4, pp. 36-39, April 1972.
8. Pyke, T. N., "Computer Networking Technology: A State of the Art Review," *Computer*, Vol. 6, No. 8, pp. 13-19, August 1973.
9. Kleinrock, L., "Survey of Analytic Methods for Computer Network Design," in *Computer-Communication Networks*, edited by F. F. Kuo, Prentice-Hall, Englewood Cliffs, 1973, Chapter 4, Section 2.
10. Fultz, G. L. and L. Kleinrock, "Adaptive Routing Techniques for Store-And-Forward Computer Communication Networks," *Proc. International Conference on Communications*, pp. 39/1-8, Montreal, Canada, June 1971.
11. Chou, W. and H. Frank, "Routing Strategies for Computer Network Design," *Symposium on Computer Communication Networks and Teletraffic*, Polytechnic Institude of Brooklyn, Vol. 22, pp. 301-309, April 1972.
12. Gerla, M., "Deterministic and Adaptive Routing Policies in Packet-Switch Computer Networks," *Data Communication Symposium*, pp. 23-28, St. Petersburg, Florida, November 1973.
13. Metcalfe, R. M., "Strategies for Interprocess Communication in a Distributed Computing System," *Symposium on Computer-Communication-Network and Teletraffic*, Polytechnic Institute of Brooklyn, Vol. 22, pp. 519-525, April 1972.
14. Haberman, A. N., "Synchronization of Communicating Process," *Comm. of the ACM*, Vol. 15, No. 3, pp. 171-176, March 1972.
15. Wecker, S., "A Design for a Multiple Processor Operating Environment," *Comp. Conf. 73, Digest of Papers*, IEEE, pp. 143-146, New York, February 1973.
16. Ravidran, V. K. and T. Thomas, "Characterization of Multiple Microprocessor Networks," *Comp Conf 73, Digest of Papers*, IEEE, pp. 133-137, New York, February 1973.
17. Jordan, B. W. and E. L. Baatz, "C.mup—North Western University Multimicrocomputer Network," *Proc. 1974 Symposium on Computer Networks*, Gaithersburg, Maryland, May 1974. Also, available from IEEE, New York (74CH0835-9C).
18. Fuller, S. H., D. P. Siewiorek and R. J. Swan, "Computer Modules: An Architecture for Large Digital Modules," *Proc. First Symposium on Computer Architecture*, Computer Architecture News, Vol. 2, No. 4, December 1973.
19. Heart, F. E., S. M. Orstein, W. R. Crowther and W. B. Barker, "A New Minicomputer/Multiprocessor for the ARPA Network," FJCC, *AFIPS Conf. Proc.*, Vol. 42, pp. 529-537, 1973.
20. Foster, C. C., "A View of Computer Architecture," *Comm. ACM*, Vol. 15, No. 7, pp. 557-565, July 1972.
21. Spier, M. J., T. N. Hastings and D. N. Cutler, "An Experimental Implementation of the Kernel/Domain Architecture," *Proc. 4th ACM SOSP*, pp. 8-21, October 1973.
22. Glorioso, R. M. and F. C. Colon, "Cybernetic Control of Computer Networks," *Modeling and Simulation*, Vol. 5, No. 2, Proc. Fifth Annual Pittsburgh Conference, pp. 819-824, April 1974.
23. Glorioso, R. M., D. W. Li and F. C. Colon, *A Schema for a Multiple-Microprocessor System*, Report DEC-3, University of Massachusetts, Amherst, February 1975.
24. Coffman, E. G., M. J. Elphick and A. Shoshani, "System Deadlocks," *ACM Computing Surveys*, Vol. 3, No. 2, pp. 67-78, June 1971.
25. Holt, R. C., "Some Deadlock Properties of Computer Systems," *ACM Computing Surveys*, Vol. 4, No. 3, pp. 179-196, September 1972.

The CERF computer system

by NEIL WILHELM, DAVID PESSEL and CHARLES MERRIAM
University of Rochester
Rochester, New York

ABSTRACT

Multiprocessor systems are becoming increasingly popular because of the increased throughput possible and the possibility of system availability despite the failure of some of the processing units. This paper describes an innovative concept in multiprocessor system design, and suggests some of the research areas which have not yet been resolved but which can be studied on this system. The system architecture is patterned after Control Data Corporation's peripheral processor "barrel" except that it possesses much more powerful functional capabilities. In addition, high speed minicomputers were used to handle all of the system I/O requirements. Research areas for which this system is particularly appropriate include computer architecture-operating systems tradeoffs and multiprocessor operating systems design and implementation. The hardware system is entirely microprogrammable, allowing for increased flexibility in evaluating various research strategies.

INTRODUCTION

Two major questions present themselves in the design of multiprocessor computer systems:

(1) How can operating systems be designed to efficiently handle processor scheduling, memory management and file and I/O management on a multiprocessor system? An important aspect of this question concerns the management of the operating system itself: should it be executed by one dedicated processor or should its functions be distributed throughout the system in some fashion?

(2) What are the appropriate architecture and operating systems tradeoffs? Because of reduced hardware costs, it is becoming increasingly possible to implement many ramifications of this issue including the general ability to modify hardware implementations once they are completed without a major cost to the system.

We plan to investigate these questions, with the assistance of a unique tool: our Computer Engineering Research Facility (CERF) Computer System. This computer system, and some of our research objectives, are described on the following pages.

SYSTEM DESCRIPTION

The CERF Computer System is depicted in Figure 1. The primary elements of the system are the four Central Processing Units (CPU's). These are independent, microprogrammable processors, featuring 64 bit data paths and 72 bit microinstructions. Each processor has its own scratchpad of 64 (64-bit) registers, although microinstructions are fetched from a common (and expandable) 1024 storage. Each microinstruction features field extraction and branch capabilities, automatic stacking of subroutine return addresses to a depth of 16, indirect register references with no time penalty, and a large selection of arithmetic and logical operators. The processor clock is 10 MHz, providing 400 ns microinstruction execution times (all microinstructions take the same time).

The CPU's communicate to the external world *via* two buses, the Stunt Box Bus, and the Memory Bus. The Stunt Box Bus is used for the attachment of specialized hardware, such as multipliers, dividers, target machine instruction decoders, etc. The Memory Bus connects the CPU's to the main memory, which initially will be 128K bytes, arranged as 64-bit words, of solid state memory. Error detection and correction will be provided by an 8 bit Hamming code appended to each memory word.

I/O devices are divided into two categories, high speed and low speed. High speed devices, such as the disc (an 80 megabyte moving-head disc) or drum (a one megabyte fixed-head disc), are each connected to the I/O Bus *via* two Selector Channels. These channels are extremely fast, special purpose, programmable computers with 16-bit words.

Low speed devices are connected through an Interdata 7/16 minicomputer with 16k bytes of 1000ns primary memory. The card reader, printer and other miscellaneous peripherals are attached to the 7/16 by a special interface called the Multiplexor Channel. This channel provides these peripherals with direct access to the 7/16's memory on a time-multiplexed basis. The CPU console is simulated by an interface to the Multiplexor Channel. Terminals and communication equipment are connected to the 7/16 by the Terminal Controller, which is a very fast programmable computer similar to the Selector Channels. The Terminal Controller also provides the path between the 7/16 and the primary memory of the CPU.

In addition to handling the low-speed peripherals, the 7/16 assists in debugging the hardware *via* the simulated console, and, with its core memory, is a convenient means of bootstrapping the system.

CERF CPU description

The need to have multiprocessing capability with three or more processors poses a number of potential design problems, primarily problems with priorities, interference, and processor lock-out. In addition, having multiple processors generally means having multiple copies of the same hardware. All of these problems can be solved by a design technique similar to one used by Control Data Corporation[1] for the peripheral processors in their 6000 and 7000 series systems. The CERF central processor hardware is divided into four disjoint subsets, corresponding to four stages of microinstruction execution. Thus a processor, at any stage of a microinstruction's execution, needs only one of the four parts of the hardware, so that *four processors can share the same hardware* provided each is in a different stage of microinstruction execution. Such an arrangement, with several processors cycling through the same hardware, is often called a "barrel".

Figure 2 shows the data flow paths of the four processors. Each processor has its own bank of 64 registers of 64 bits each; these are the only elements which are not shared among the processors. The functional elements, which operate on data, consist of the field extractor, which can extract any field of contiguous bits from a word, the field depositor, which can deposit an arbitrary length field into a word, and the arithmetic-logic unit (ALU) which can perform the usual arithmetic and logic operations plus a number of specialized ones. Note that the B-bus, which provides one of the two operands required by the ALU and field depositor, is fed by the field extractor, so that one of the operands could in fact come from any field of a word in a register.

An important issue which must be considered is whether or not the operations shown can be done with reasonable dispatch, so that the processors are fairly fast. Calculations, using manufacturer's worst-case propagation delay specifications, show that a staging time of less than 100 nanoseconds, i.e., a clock rate of 10 MHz, is feasible. This yields a microinstruction time per processor of 400 ns which, from the rule-of-thumb that 10-30 microinstructions are required for each target machine instruction, implies a target machine instruction execution time of 4 to 12 micro-

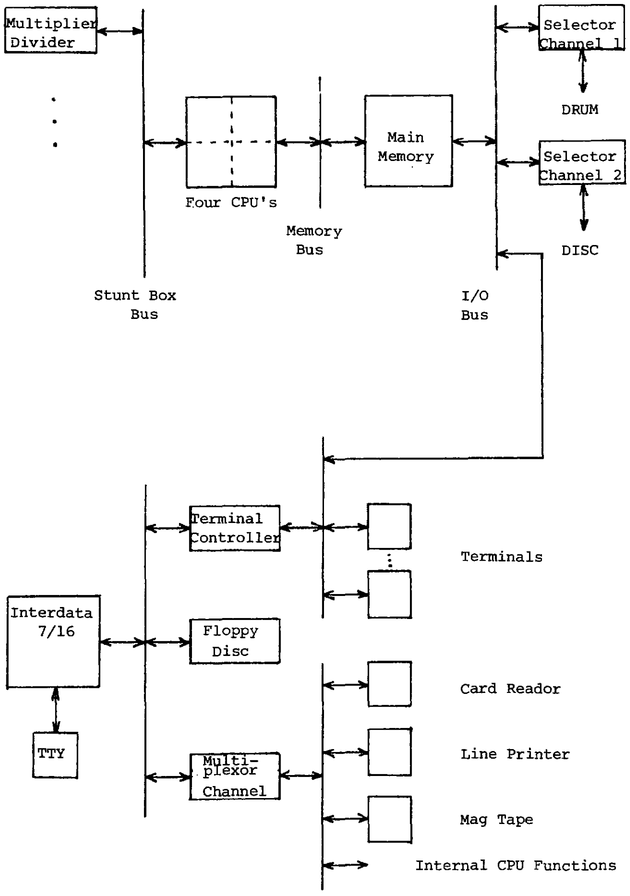


Figure 1—Computer Engineering Research Facility (CERF) computing system

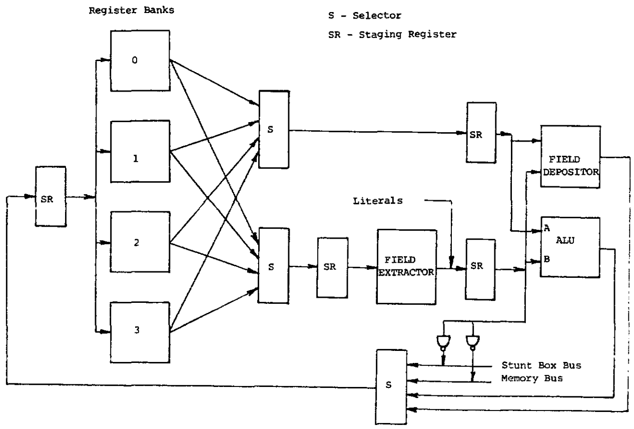


Figure 2—CERF computer system data flow

seconds on *each* processor. The *overall* execution rate would be 10^7 microinstructions per second, and .3 to 1 million target machine instructions per second.

It is essential that the microinstruction set be adaptable and powerful. To understand what this implies in terms of microinstructions, one must first realize what consumes most of the effort of emulators: decoding the instructions of the target machine. To reduce this overhead the set of microinstructions must include ones for extracting fields of bits from words and for making conditional branches and subroutine calls. All of these objectives are met with the CERF CPU microinstructions.

The design of CERF is presently complete, and it is anticipated that its construction will be complete by June, 1976. Its design has been made in full cognizance of and with careful consideration for the research areas described here. CERF will provide a unique research vehicle, and has been configured in such a way so as to provide a general purpose timesharing system to the University community. This user community is necessary to provide a load on the system during the testing of research problems.

We now describe some aspects of the hardware implementation.

HARDWARE IMPLEMENTATION

The CPU's

The Schottky TTL logic family was selected for the implementation of the CPU's. Emitter-coupled logic (of the ECL 10K family) was considered, but it was rejected because of power consumption, wiring (particularly the need for pulldowns), and interfacing problems (requires level translators to mate with other logic families), because of a shortage of MSI and LSI function as compared with TTL, and because of its high cost. An ECL or ECL-TTL hybrid system could easily run twice as fast.

Recent advances in bipolar memory technology not only reduced circuit complexity over preliminary estimates, but also reduced system cost. Processor register banks are implemented with 64×9 RAM's (Fairchild 93419) which provide a worst-case access time of 50ns (actual measurements with a lGHz sampling system show an address-to-output access time of about 30ns). Thus the entire set of four banks of 64×64 registers requires only 32 integrated circuits. The microstore is constructed from 1024×1 bipolar RAM's (Fairchild 93415) with a worst-case access time of 70ns (measurements show a typical access time of about 37 ns), requiring only 72 integrated circuits for a 1024 word storage.

The primary limiting factor in CPU speed is the delay in the ALU-test-branch decision path. Because branch options on every microinstruction are desirable, and because branching should carry no time penalty (since it is done so frequently in microprograms), this path is crucial to the machine's performance. The only solution is to calculate *both* of the possible successors to each microinstruction, and select the correct one at the last possible instant.

Main memory

Dynamic MOS RAM's were the choice for the main memory, because of their low cost, availability, and reliability. Memory system reliability is enhanced by using an 8 bit Hamming code, which provides single error correction and double error detection, on each 64 bit datum. Thus memory words are 72 bits in length.

Four-way interleaving is used to increase memory bandwidth which, for an approximate cycle time of 500 ns, is 8 megawords or 64 megabytes per second. The useable bandwidth is somewhat less than this. New memory accesses can be initiated while others are in progress, thereby reducing idle time.

Selector channels

The selector channels are implemented with 4-bit slice processor elements (AMD 2901), being essentially very fast 16 bit minicomputers. Each has typically 1024 words of bipolar memory (the same as the microstore) for programs and data. Special "vector" instructions are used to handle the extremely high data rates of the disc drives. Special functions, such as error correcting coding, are implemented in hardware.

RESEARCH GOALS

Most conventional operating systems are almost exclusively oriented towards the concept of single processor systems. Although these operating systems may allow multiprogramming, the ramifications of multiprocessing are generally avoided. Multiprocessing systems have been examined in limited detail elsewhere.[2, 3] This study will consider in depth various aspects of multiprocessor operating systems. In particular, we will study whether one operating system can effectively and efficiently coordinate the activities of more than one processor in a multiprocessor system. In a multiprocessor system each processor may have an independent address space mapped onto one physical memory, or independent memories may be available to each processor. In either case, the operating system is now faced with the task of coordinating various memory maps. Similarly, each processor in a multiprocessor system may execute programs from independent job streams or there may be only one job stream being scheduled onto all of the processors. In both cases, the operating system must handle scheduling of a far more complex form than on a uniprocessor system. Finally, a separate file and I/O system may exist for each processor. This results in yet another set of operations more complex than on conventional systems. Various problems may arise as a direct result of this type of system. Among these are: (1) primary memory contention: since more than one simultaneously active processor and task can address the same memory module, techniques must be explored to reduce or avoid memory contention at that module, and (2) file protection: adequate file protection is complicated because of the existence of simultaneously active processors and tasks which may access the same files. File structures which insure data integrity must be developed and implemented.

A second area to be considered concerns architecture operating systems trade-offs. An operating system is very dependent upon the architecture (i.e., the instruction set, registers, etc.) of the processor on which it is run. This relationship is so close that not only the effectiveness and efficiency of the operating system but the very *structure* of the operating system is determined by the processor architecture. For example, the "cactus stack" design of the B-6700[4] leads naturally to the tree-like hierarchy of processes used by the operating system. Of course one can always *impose* an operating system design on an unsuitable processor, at the risk of losing efficiency.

We can delineate three basic areas in which the operating system-processor interaction is crucial. These are processor scheduling, memory management and file and I/O management. For each of these areas, trade-offs can be made between performing important functions as "hardware macro's" (e.g. as special processor instructions or using dedicated processors) or as "software macro's" (i.e., aggregates of primitive hardware instructions). We can increase system speed by using more "hardware macro's" in exchange for greater processor cost and complexity and reduced system flexibility. On the other hand, we can reduce system cost and increase flexibility by using "software macro's", but at a reduction in efficiency.

Existing systems are examples of *a priori* decisions regarding the distribution of functions between hardware and software. For the IBM S/360,[5] essentially all operating system functions are implemented without hardware assistance. On the other hand, the Berkeley Computer Corp. BCC-500[6] (now at the University of Hawaii) features specialized processors performing such functions as scheduling and memory management.

We propose to investigate, theoretically and empirically, several specific areas of operating system-architecture tradeoffs, using the CERF multiprocessor computer system as a tool. One of these areas is the design of operating systems for multiprocessors, emphasizing the synchronization primitives and the architectural features necessary to support them. Hardware provisions for mutual exclusion are especially important in a multiprocessor system, since the simple expedient of turning off the interrupts and simulating mutual exclusion primitives in system software works only for a uniprocessor system. Since P and V operations may imply changes in processor scheduling, the relationship between the hardware and software is very close at this point.

Another important consideration is the distribution of the operating system functions to various system processors. Should important functions such as processor scheduling be shared among the central processors (as in the original plan of the CMU Hydra system), should one of the central processors be dedicated to these operations, with the other processors as slaves, or should these functions be distributed to more specialized "peripheral processors" (as done in the BCC-500 and the CDC 6000 and 7000 series)? The CERF system is ideally suited for this type of investigation. One could write special microcode for one of the central processors, making it the master. On the other hand, the drum controller can be reprogrammed to handle the memory management, or the Interdata 7/16 could be programmed to perform processor scheduling.

Our current plan, now under way, is to develop an initial target architecture which is "extensible" so that we may add the features we need with relative ease, and is also well-structured, so that ALGOL-like languages can be fully-supported and compiled efficiently. Because the basic machine architecture will remain relatively fixed, we can expect to get meaningful performance comparisons as we explore the various possibilities.

CONCLUSION

We have described an innovative approach to multiprocessor system architecture, yielding greatly increased hardware flexibility and applicability at only a slight increase in cost. This system provides a unique vehicle for research in multiprocessor architecture and operating systems. The research objectives have been briefly described, and the results of these studies will be presented in future publications.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of the College of Engineering and Applied Science in funding this project.

REFERENCES

1. Thornton, J. E., "Parallel Operation in the Control Data 6600," *AFIPS SJCC*, Volume 24, 1964.
2. Wulf, W. A. and C. G. Bell, "C. mmp—A Multi-Mini Processor," *AFIPS FJCC*, Volume 41, 1972.
3. Baskin, H. B., B. R. Borgerson and R. Roberts, "PRIME—A Modular Architecture for Terminal-Oriented Systems," *AFIPS SJCC*, Volume 40, 1972.
4. Organic, E. I., *Computer System Organization: The B5700/B6700 Series*, Academic Press, New York, 1973.
5. IBM, *IBM System/360 Principles of Operation*, File No. S360-01, November 1970.
6. Wall, C. F., *Design Features of the BCC 500 CPU*, TR R-1, January 3, 1974.

A parallel processor for evaluation studies

by GARY J. NUTT
University of Colorado
Boulder, Colorado

ABSTRACT

The Multi Associative Processor system is a multiple control unit parallel processor capable of executing a maximum of 8 single-instruction-stream, multiple-data-stream programs simultaneously. The architecture supports parallelism at two levels: the lower level is the tightly coupled parallelism typical of array processors, and the higher level is the more loosely coupled parallelism between independent processes. The architecture of the machine is described and an example program for the machine is given to illustrate many of the concepts of the architecture. Measurement and evaluation studies on the machine are also briefly discussed.

INTRODUCTION

The needs of present day, large scale computer users have led to the development of a variety of parallel processor systems. The exploitation of parallelism becomes necessary for these users since electronic signal propagation has become a limiting factor to the speed with which certain computation can be carried out on a sequential processor.[2] There exist a number of architectural families that exploit parallelism in one way or another—pipeline processors, vector machines, multiprocessors, array processors, and associative processors.[6] The inherent complexity of these processors magnifies the need for measurement and evaluation techniques to design and tune the systems. The design of a system includes the organization of the software as well as the hardware; performance evaluation has often been shown to be important in the design phase of sequential processor systems, e.g. see Reference 5. Past experience has also pointed out the need for evaluation techniques in tuning sequential processor systems, e.g. see Reference 9. Since the parallel processors must include more complex hardware and software, the need for sound measurement and evaluation techniques is even more important than it was for the third generation sequential processor systems.

In order to investigate design and evaluation techniques for any given class of computer systems, one must either employ a prototype system, or construct a model to represent members of the class. The ultimate test for any hypothesis is to carry out experiments with the real subject; in the case of measurement investigations for computer systems the ultimate test is to try techniques on a physical computer. There are, however, certain advantages to experimenting with models rather than the real system: economics is an obvious advantage. Other advantages include the ability of the model to ignore certain factors in the analysis. It is also easy to try various designs and configurations with a model, where real systems are not only expensive to modify but also time consuming to test. Extensive modeling should precede prototype testing.

In this paper a hypothetical associative array processor system is described, where the primary intent for introducing another "paper" machine is not only to display a novel architecture, but also to provide a model to be used for investigating measurement and evaluation techniques for parallel processors. To help explain the design concepts of the machine, a sample program is provided, following the architecture description. The paper serves to document the system model, to display the current state of work on the project, and to describe future work.

THE ARCHITECTURE OF THE SYSTEM

The medium of this study is called a Multi Associative Processor (MAP) computer system. The machine is a member of the class of single-instruction-stream, multiple-data-stream (SIMD) parallel processors.[6] These computers employ a control unit to decode a string of instructions stored in a main memory and to cause a collection of arithmetic units to execute the instructions on data stored in private memories associated with each arithmetic unit. Some prominent members of this class of computer systems include the ILLIAC IV,[3] the PEPE system,[7] and the Goodyear STARAN.[13] The MAP system shares many properties of these systems, but also has others unique to its architecture.

Figure 1 is a block diagram of the system, and the

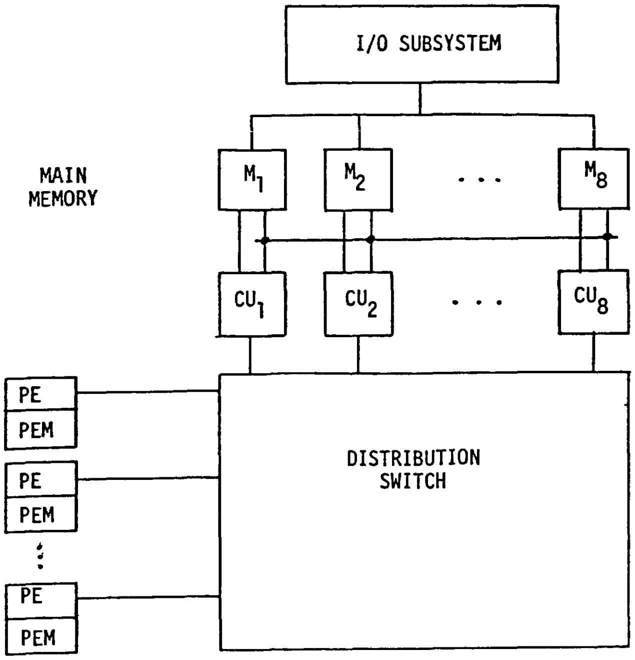


Figure 1—MAP block diagram

components are first briefly described with more detailed description given in subsequent subsections. (An even more detailed description of the components is given in Reference 1). The most striking difference between MAP and the production systems mentioned above is the existence of eight control units rather than one. Each control unit (CU) along with a subset of the arithmetic units, (called processing elements or PEs), corresponds to a SIMD architecture; MAP supports eight SIMD programs simultaneously. Another prominent difference is the absence of a "host processor" to execute system software, e.g., the operating system, compilers, etc. All system software is executed on the MAP by a CU and one or more PEs. The main memory system incorporates eight memory modules interconnected by a bus connected to the I/O system and by a bus connected to each control unit; each memory module (MM) is preferred by one CU and, normally, the instruction stream for CU_i is stored in MM_i.* The distribution switch shown in the figure routes commands and data from a CU to the subset of PEs currently allocated to that CU, and between PEs. The I/O subsystem is one of those portions of the system for which no explicit design has been considered. It is assumed to be a conventional system that handles peripheral devices and provides for the transmission of data to and from the main memory.

The control unit organization

The primary purpose of each CU is to implement the concept of a process. Each process that is implemented on a CU is assumed to be decomposed into a set of identical tasks which can be executed in parallel by a set of PEs. Other instructions decoded by the CU, such as branching instructions, are executed directly in the CU without referencing the set of PEs. The CU must also implement interprocess communication between processes on different CUs.

Figure 2 is a conceptual block diagram of a control unit. The main memory interface consists of three registers: the Data Register is used to transmit data words to/from the main memory, the Address Register is used to send CU-generated addresses to the main memory address register, and the Instruction Word Register holds an instruction word from main memory that is to be decoded and executed. It would be possible to replace the Instruction Word Register by a look-ahead buffer to decrease CU idle time while waiting for the next instruction to be read from main memory. (Current work on this approach is in progress.) The Index Register, CX, allows indexed addressing of the main memory by the CU.

The CU Instruction Processor executes those instructions that are not broadcast to PEs for execution. The functions implemented by this processor include conditional and unconditional branching, (using CX and the Compare Register, CR); and CU intercommunication instructions, (using the 8-bit Signal Register and the 8-bit ID Register). Most conditional branches are determined by inspecting the relationship between contents of CR and CX, or only the contents of the main memory index register, CX. The use of Signal and ID in interprocess communication is discussed in a later portion of the paper.

The Data Input (Output) Broadcast Register is used to send (receive) data from the currently allocated set of PEs via the distribution switch. Loading or unloading of the registers is controlled by the Instruction Processor under the supervision of the Instruction Decode Unit.

The Instruction Decode Unit invokes the Instruction Processor for the execution of CU-executable instructions, and generates the sequence of microinstructions that are to be executed by PEs. The 6-bit Instruction Broadcast Register is used to put a single microinstruction into the distribution switch, which subsequently routes the microinstruction to the set of PEs currently allocated to the given CU for execution. The decode unit must handle all timing considerations for broadcasting microinstructions and data over the distribution switch. It must also coordinate its operation with that of the distribution switch.

* If a distributed operating system is employed in MAP, then the program for CU_i may not be stored in MM_i, but may either be distributed over the modules, or a ninth module for operating system code and data might be incorporated.

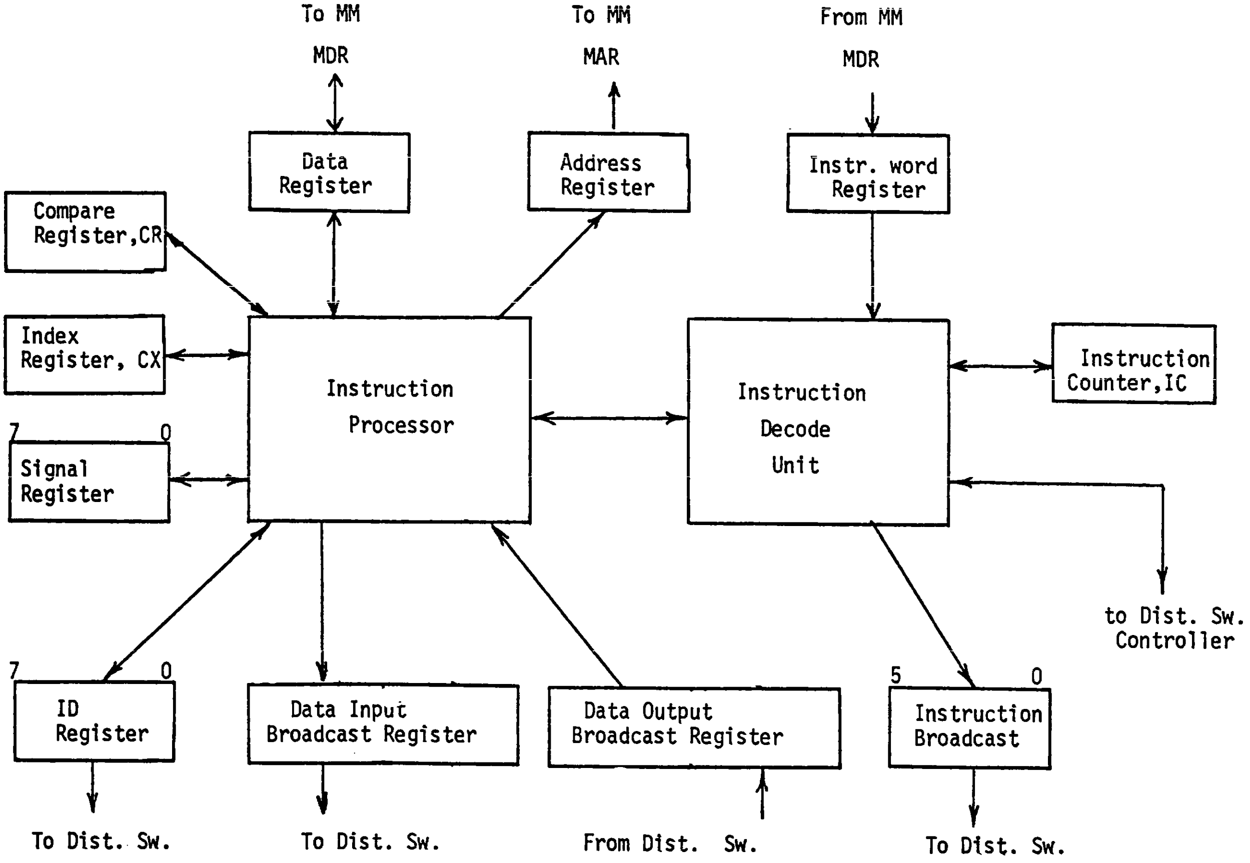


Figure 2—Control unit organization

The processing element organization

PEs correspond to the arithmetic/logical unit of a conventional computer system. Their purpose is to carry out a particular microinstruction execution on multiple data items in parallel with one another. All PEs are treated as identical resources, and conceptually, there is no addressing relationship between any two PEs; no linear relationship exists between PEs as it does in the ILLIAC IV.[3]

Figure 3 is a block diagram of the PE organization. The dynamic connection of PEs and CUs takes place via the distribution switch discussed in the next subsection, and the interface to this bus system takes place at the Instruction Selection Unit, the Input (Output) Data Register, IDR (ODR), and the Input (Output) Control Register, ICTL (OCTL). The Instruction Selection Unit contains an 8-bit Owner Register; when a PE is allocated to a CU, the owner register of the PE is set to match the ID Register of the CU. Each microinstruction broadcast by a CU is tagged with the ID Register content, and the Instruction Selection Unit compares ID and Owner Registers to identify microinstructions intended for the given PE. The microinstruction is then passed on to the Associative Unit of the PE.

The architecture of a SIMD processor allows all PEs allocated to a CU to simultaneously execute a microinstruction, but in most programs, it is necessary to be able to selectively deactivate some of the PEs during the execution of some code segments. For example, a program may be organized such that each element of an n dimensional vector is loaded into a distinct PE memory and that the PEs should perform a computation sequence on only those coordinates in the vector that are greater than zero. The code sequence would cause each coordinate to be loaded into a PE register from the respective PE memory, and then only those PEs whose register content is greater than zero would continue to execute microinstructions, while the remaining PEs would remain idle. The Associative Unit is used to control this selective activation/deactivation on the basis of conditions that exist in a PE at any given time. PE selection status is determined by the contents of the programmable 8-bit Key, Mask, and Select Register within the Associative Unit as follows: If Mask bit i is set then Key bit i is compared with Select bit i, and if all such bits have the same value,

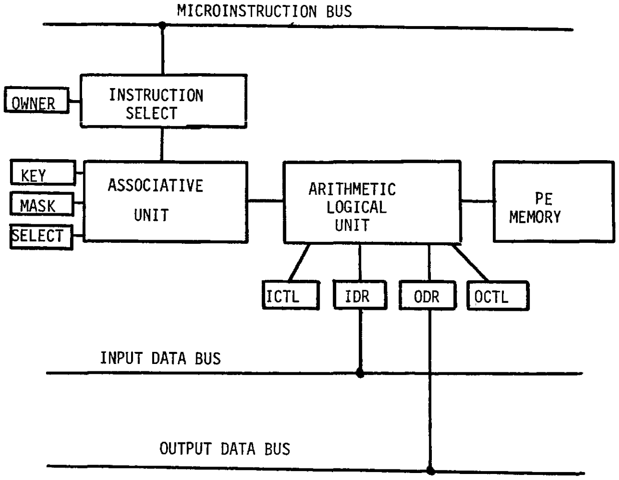


Figure 3—PE organization

the PE status is active. In the case that Mask bit i is reset, the Key and Select bits are ignored. For example, if Key=10101010, Mask=00111100, and Select=01101011, then the PE is active; if the same Key and Select are used with Mask=00001111, the PE is deactivated because the least significant bits of Key and Select do not match. This approach allows up to eight independent activation conditions to be stored in the registers, implying that extensive subsetting of PE conditions can be saved without recomputing those conditions. All three registers can also be stored/loaded to/from the respective PE memories if more than eight activation conditions are to be used in a code segment.

The Associative Unit modifies Key and Mask whenever an associative instruction, (i.e., microinstruction sequence) is broadcast to redefine conditions under which PEs allocated to the CU should be activated or deactivated. The Select register contents are determined by a different set of associative instructions that are related to PE arithmetic register contents. Examples of the use of the registers are given in a subsequent sample program segment.

The Arithmetic/Logical Unit is a small processing unit with a double-length accumulator and a PE index register. Operands for the ALU may be stored in the PE Memory (accessed with the PE index register), or may be global operands loaded from (stored into) the IDR (ODR). The ALU executes floating point and integer arithmetic instructions, and logical instructions.

The interchange of data between PEs and a CU over the distribution switch uses the ICTL and OCTL in conjunction with the IDR and ODR respectively. A discussion of their use appears in the next subsection.

The distribution switch organization

The distribution switch is used to route microinstructions from a CU to an arbitrary subset of the PEs, to route data between a CU and an arbitrary subset of PEs, and to route data between individual PEs.

The mechanism for routing 6-bit microinstructions is a crossbar switch, where crosspoint connections are determined by ID and Owner registers in CUs and PEs. The connection mechanism precludes a requirement for resolution and queuing mechanisms at each crosspoint. It also disallows any form of simultaneous PE sharing by two or more CUs. This approach has been taken since CUs are expected to broadcast microinstructions at a high rate—on the order of one microinstruction per 100 ns.

Data transfers occur much more infrequently—previous MAP interpreter monitor results indicate that data (or a PE memory addresses) are broadcast about once every six microinstruction cycles.[11] Since the data bus is expected to be much wider than the microinstruction bus, a combination of bus sharing and a crossbar switch has been employed in the design. Figure 4 is a diagram of the data bus organization; it consists of an $8 \times k$ crossbar switch where $8 \leq k \leq 12$. Each of the k buses, called *bus sectors*, are shared by approximately 100 PEs for data transmission. In a previous study, it was shown that this configuration does not introduce appreciable performance degradation due to bus conflicts.[11] PEs from a given sector may be allocated to any CU, but sector conflicts are minimized if the number of CUs owning PEs in any given sector is minimized. In another study, some PE allocation algorithms that reduce sector sharing among CUs were investigated, and cost-effective algorithms for the sector organization were determined.[12]

Data is transferred over the bus system using the ICTL and OCTL registers mentioned in the PE description. Input and output operations are symmetric, and so only the input operation is described. The ICTL is initially loaded with a value representing a delay time. For each cycle to which the CU controlling the operation has access to the data bus, ICTL is decremented by one. When the ICTL reaches zero, the data currently on the input bus sector for the PE is gated into the IDR of the PE. By setting the ICTL registers of PEs to different values, it is possible to rapidly transfer an entire data stream from the main memory into a set of distinct PEs; the ICTL and OCTL can also be used to transfer data from one PE to another via the bus system and the data registers in a common CU. In the case that two or more PEs have identical ICTL contents, all such PEs retrieve data from the input data bus, if two or more PEs have identical OCTL contents, the data placed on the output data bus is the logical OR of all ODRs.

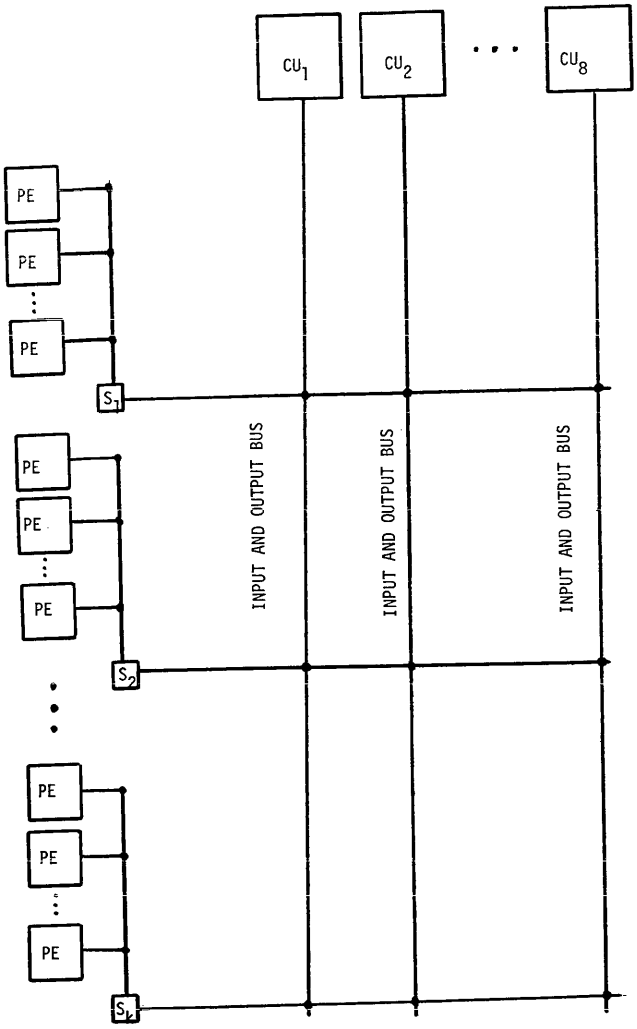


Figure 4—The distribution switch

The main memory subsystem

The main memory subsystem can be accessed by the I/O subsystem or the set of CUs, as indicated in Figure 1. Each main memory module is preferred by a particular CU, although any CU can access any module. The rationale for this organization is to reduce memory conflicts due to CU access. Previous work indicates that this main memory organization using core technology is an acceptable design, for most applications.[11]

Control unit intercommunication

The mechanism for allowing communication among the various control units is open to revision and refinement as experience in the design of an operating system progresses. The mechanism should allow a variety of system implementations ranging from a basic operating system that is executed on a single CU which supports only uniprogramming on the remaining CUs, to a more sophisticated distributed system that runs on any CU, perhaps in parallel with itself. This latter design strategy would also support multiprogramming of all CUs.

The process identifier is the ID register content. Let $ID_K(i)$ denote the K^{th} bit of the ID register in CU number i. If $ID_K(i) \wedge ID_K(j) > 0$ for some bit K, then CU_i can *cooperatively communicate* with CU_j, and vice versa; otherwise, no communication is possible between the two control units. Suppose that in addition, $ID_K(j) \leq ID_K(i)$ for all K. Then CU_i can communicate with CU_j in a *privileged manner*, (it is not allowed for two CUs to have exactly identical ID registers); privileged communication implies cooperative communication.

Each CU contains three 1-bit registers to aid in communication: ARM_i is used to disable any privileged interrupt directed to CU_i if $ARM_i = $ *false*. Similarly, $ABLE_i$ is set to *false* if CU_i cannot accept an interrupt in a cooperative manner. The ACK bit is used to acknowledge the receipt of an interrupt. An interrupt register contains an address to be branched to when a cooperative interrupt is received. The Signal register is used to designate the CU that is to receive a message.

Only some of the CU intercommunication instructions can be executed successfully by CU_i if CU_i has privilege with respect to CU_j. The PREEMPT instruction allows CU_i to interrupt CU_j provided that $ARM_j =$ *true;* otherwise, the interrupt is refused, and $ACK_i =$ *false.* (CU_j cannot disarm itself.) The preemption results in CU_j being disarmed to prevent other interruptions from occurring, CU_j finishing its current machine instruction execution, saving the instruction counter in main memory module j at location zero, and restarting CU_j on another code segment after passing an address through the CX registers. CU_i then continues execution.

Once CU_j has been preempted by CU_i, CU_i can alter the ID register of CU_j; the new ID register content is determined under the following restriction:

$$ID_K(j) \leftarrow ID_K(i) - y$$

where

$$0 \leq y \leq ID_K(i)$$

i.e., CU_i can define any new ID register content for CU_j (or itself) as long as no bit in ID of CU_j is set when the corresponding bit in CU_i is reset. This prevents a CU from increasing its communicative power either directly or indirectly. Whenever a preempting CU returns a preempted CU to its state preceding the interruption, it must restore the state of the interrupted CU.

The above discussion was concerned with communication in a privileged manner; cooperative communication is implemented as follows:

When a SIGNAL instruction is executed by CU_j with $ID_j \equiv Signal_i$, an interrupt is passed to CU_j provided that ARM_j is *true* and $ABLE_j$ is *true.* The interrupt activity includes temporarily preventing a PREEMPT, and disabling cooperative interruption. The current instruction count of the signaled CU_j is saved in memory module j at location 1. The address of the next instruction is then obtained from the interrupt address register. CU_j is then armed for privileged communication and remains disabled for cooperative communication until an ENABLE instruction is executed by CU_j. A BLOCK command can be used after a SIGNAL to wait for a response from the signaled CU.

If a CU is executing in disabled mode, then any signal directed to the process might be queued on the process descriptor that has an ID content matching the signaling CU signal register. No hardware provision is made for actually appending signals to process descriptors of processes not currently running on a CU, nor for handling signals to nonexistent processes since these are software tasks of individual operating systems. The operating system design must handle such situations, perhaps in a manner similar to the message buffering scheme used in the RC 4000 multiprogramming scheme.[4]

SAMPLE MAP PROGRAMS

Previous machines have often been aimed at particular application areas, e.g., the Illiac IV is primarily intended for floating point matrix computations, while PEPE and STARAN have often been used to do nonnumeric processing. MAP is an attempt at a design appropriate for both areas, although it is weak in its I/O capability. To illustrate the design philosophy, a brief description of a MAP algorithm is now presented.

A typical numerical algorithm employed in matrix computation is the Gaussian Elimination Method for solving a system of linear equations. The method can be efficiently implemented on MAP as the Gauss-Jordan method with full pivoting. It is assumed that the coefficient matrix and column vector of the right sides of the equations have been stored in the main memory module of a CU. The method employs n+1 PEs for an $n \times n$ system of equations, and each PE is loaded with one column of the coefficient matrix: the right side column vector is loaded into the n+1st PE. PE memory loading is accomplished by the following code segment (stated in prose form rather than as mnemonic MAP instructions):

1. Clear all Select Registers.
2. Deactivate all PEs but one and set its Select bit 0 (bit 0 corresponds to the condition that the PE is being loaded).
3. Load a column vector from main memory using CX to index main memory, PX to index PE memory.
4. Reset Select bit 0 and set Select bit 1. (Bit 1 indicates that the PE is loaded.)
5. Activate all PEs such that Select bit 1 is reset.
6. If at least one PE is active, go to step 2.

This code segment requires about 10-15 MAP instructions, and is executed n+1 times.*

In the next part of the program, the elimination algorithm is implemented. Select bit 0 is used to save various temporary conditions in searching for a pivot element, bit 1 marks the PE containing the maximal element in any row of the coefficient matrix, bit 2 marks columns to the right of the column being reduced, bit 3 marks the PE containing the columns currently being reduced, bit 4 indicates that a column has already been reduced, and bit 5 marks the PE containing the right side vector. The algorithm requires n loops through code that determines the largest coefficient in a row, and after determining a maximal coefficient, row transformation is accomplished by activating all PEs and executing two indexed loads and stores on the two rows to be interchanged. Column transformation is implemented by exchanging column number identifiers stored in the PEs and requires only 4 instructions. (A copy of the original column numbers must also be saved in each PE in order to trace the column transformations performed during the execution of the algorithm.) The elimination process requires n loops through code containing a division, a multiplication, and a subtraction, where each operation is applied to all columns (PEs). The order of time for executing this Gauss-Jordan algorithm on MAP is n^2 compared to n^3 on a sequential processor. The algorithm could be speeded up by using n^2 PEs, or by skewing matrix entries in the PE memories as is commonly done in the ILLIAC IV.[8] The order of the algorithm reduces to linear time if n^2 PEs are used, since all rows can be reduced simultaneously, and pivot elements are determined by one associative instruction.

THE EVALUATION STUDY

As stated in the introduction, MAP is being used as a medium to test evaluation and monitoring techniques for associative and array processors. The basis for much of the work is an interpreter for individual MAP programs, called MAPSIM.[10] The program represents one CU and an arbitrary number of PEs as required by a MAP program. It ignores bus and memory conflicts, but allows one to write and test individual programs for the machine. Each program can be monitored during execution, with monitor resolution at the machine instruction (*not* microinstruction) level. Using this method, a detailed description of the effect of any given job can be measured and the data can be used to drive other high level models that introduce resource com-

* The PE loading is a sequential task, but the performance can be increased by loading the PEs one row at a time rather than by columns. To do this, the ICTL registers of all PEs are used to "stream load" the rows from main memory, where only one PE is removing data from the input bus at a time, but the time between PE load operations is small.

petition due to the existence of multiple CUs. The investigations of the memory and bus system design were carried out in this manner. A drawback to this approach is that it is difficult to effectively study CU intercommunication.

Now that the design has stabilized to some degree, current evaluation studies are directed at two major areas: operating system design and user program measurements. The approach taken in the operating systems work is to first compare three families of systems. The first strategy employs a dedicated control unit for the operating system and the remaining 7 CUs are uniprogrammed. The second strategy again uses a dedicated CU for the operating systems, but implements multi-programming on the remaining CUs. The final strategy multiprograms all CUs and executes the operating system as a set of high priority processes on all CUs. A comparison of the three strategies is forthcoming.

User program measurements can be carried out directly on MAPSIM. In conventional processors, a typical measurement of a user program results in the generation of an instruction execution distribution to determine frequently-used portions of the code. This measure is appropriate for MAP programs, but additional information about PE activation/deactivation is perhaps even more important. Current and future work on MAP is concerned with the investigation of such measures, and the implied monitoring tools required to obtain such measures.

ACKNOWLEDGMENT

The National Science Foundation has supported the study of MAP under Grant number GJ-42251. The original concept of the machine came from many long discussions with Clarence Ellis; the following graduate students at the University of Colorado have participated in the subsequent development: Roger Arnold, Karl Williamson, and John Burke.

REFERENCES

1. Arnold, R. D. and G. J. Nutt, *The Architecture of a Multi-Associative Processor*, University of Colorado, Department of Computer Science, Technical Report No. CU-CS-070-75, June, 1975.
2. Baer, J. L., "A Survey of Some Theoretical Aspects of Multiprocessing," *ACM Computing Surveys*, Vol. 5, No. 1, pp. 31-80, March, 1973.
3. Barnes, G. H., et al, "The ILLIAC IV Computer," *IEEE Transactions on Computers*, Vol. C-17, No. 8, pp. 746-757, August, 1968.
4. Brinch Hansen, P., "The Nucleus of a Multiprogramming System," *Communications of the ACM*, Vol. 13, No. 4, pp. 238-241, April, 1970.
5. Campbell, D. J. and W. J. Heffner, "Measurement and Analysis of Large Operating Systems during Systems Development," *AFIPS Proceedings of the FJCC*, Vol. 33, pp. 903-914, 1968.
6. Flynn, M. J., "Some Computer Organizations and Their Effectiveness," *IEEE Transactions on Computers*, Vol. C-21, No. 9, pp. 948-960, September, 1972.
7. Githens, J. A., "A Fully Parallel Computer for Radar Data Processing," *NAECON '70 Record*, pp. 290-297, 1970.
8. Kuck, D. J., "ILLIAC IV Software and Applications Programming," *IEEE Transactions on Computers*, Vol. C-17, No. 8, pp. 758-770, August, 1968.
9. Lucas, H. C., "Performance Evaluation and Monitoring," *ACM Computing Surveys*, Vol. 3, No. 3, pp. 79-91, September, 1971.
10. Nutt, G. J., "Some Uses of Simulation in Systems Design," *Proceedings of the ACM-NBS Third Symposium on the Simulation of Computer Systems*, pp. 221-228, August, 1975.
11. Nutt, G. J., "Memory and Bus Conflict in an Array Processor," Submitted for publication.
12. Nutt, G. J., "Some Resource Allocation Policies in a Multi Associative Processor," Submitted for publication.
13. Rudolph, J. A., "A Production Implementation of an Associative Array Processor," *AFIPS Proceedings of the FJCC*, Vol. 41, pp. 229-241, 1972.

Asynchronous speed-independent arbiter in a form of a hardware control module

by H. SECHOVSKY and S. JURA
Research Institute for Mathematical Machines
Prague, Czechoslovakia

ABSTRACT

The attained parameters and the development trends of LSI circuits make possible the realization of untraditional computer structures with higher number of cooperating processor and control units or modules. The parallel processes realized in these modules are in majority mutually asynchronous and the length of duration of individual parts of these processes and their mutual time relation cannot be determined in advance. In hardware realization of such systems it is often most convenient when the individual modules behave toward each other like asynchronous speed-independent automata. For interconnection of such modules and for synchronizing the processes realized in them it is advantageous to use so-called arbiters (also called semaphores) whose formal description has been introduced in the theories of parallel processes.

The text contains a detailed description of a hardware construction of the arbiter having the character of a universal control module guaranteeing the reliable function at arbitrary time sequence of input requests. This quality is important in view of the danger of metastability so that an arbiter cannot be excluded. In the design of the circuit the simplicity and speed of function have been especially considered and the measured time parameters of the arbiter are quoted.

INTRODUCTION

If we analyze the development trends of the computer technology and the influence (especially of the parameters) of the LSI circuits on the evolution of the structure of computer systems, we have to conclude that the increasing interest in the theories of parallel processes and multiprocessor, e.g., micro-multiprocessor systems is conditioned especially by reality, that the speed of the circuits produced by the technology of the semiconductor integrated circuits is coming close to the limit value for this technology, therefore the achievement of substantially higher speeds by its further development cannot be expected. Nor in the coming years an essential change in the technology of computer circuits is to be expected, considering the facts of low costs, easy mass-production and a high reliability at the same time—the facts imply that the semiconductor technology has nowadays no serious competition. The technology thus defines the speed limits which cannot be surpassed by one-processor system. This problem is solved with multiprocessor systems and other systems with parallel processes that enable, among other things, the achievement of higher speeds.

In all such systems, it is necessary to solve the problem of synchronization of parallel processes and of allocation of shared sources e.g., memories, peripherals, registers, information buses etc. This synchronization should be solved in hardware in case the system cannot be designed in the way of synchronous automata system with common clock signal.

A very broad application in the coordination of parallel working hardware modules has the arbiter, having generally n couples of speed-independent input and output signals: $R_1, A_1 \ldots R_n, A_n$. (See Figure 1.) The function of the arbiter is outlined in Figure 1 and it can be described on Petri-net. (See Figure 2.) The token from the initial place can fire only one of the transitions controlled by input signals $R_1 \ldots R_n$. The firing of these transitions is determined partly by the time sequence of logic ones arriving to inputs $R_1 \ldots R_n$

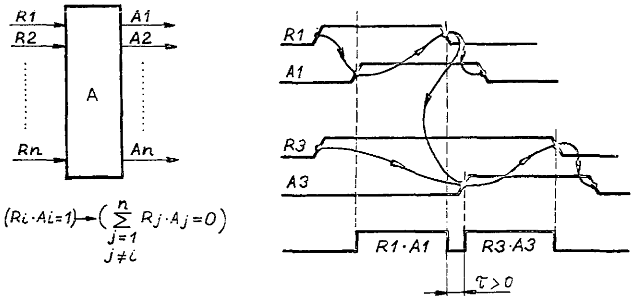

R1
R2
Rn
A
A1
A2
An
$(R_i \cdot A_i = 1) \rightarrow (\sum_{j=1, j \neq i}^{n} R_j \cdot A_j = 0)$
R1
A1
R3
A3
R1·A1
R3·A3
τ > 0

Figure 1

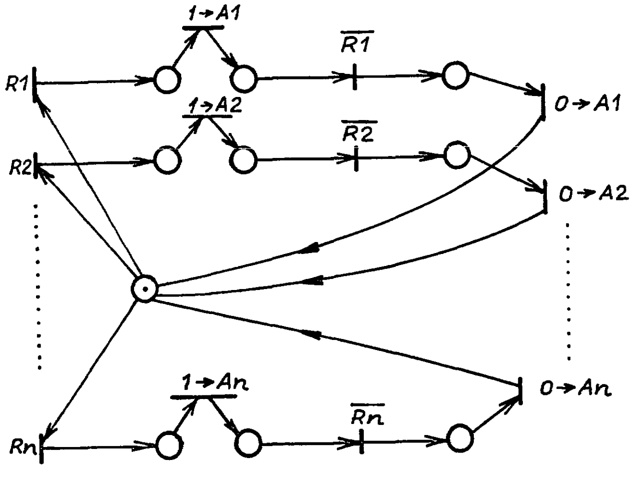


Figure 2

partly by the algorithm of the priority allocation to these requests. Under references to Petri-net in this paper, the authors have in mind so-called safe Petri-net, whose properties are described in detail in References 1, 4 and 8.

The important quality of the arbiter is the priority allocation only to one request R_i and sending out the logic one only to one output A_i. The signals R_i,A_i are in the hand-shake relation that guarantees the speed-independence of the circuit.[4] But the speed-independence of the couples R_i,A_i is the only regularity that is guaranteed for the input signals. The arbiter must work reliably in a common case when the time interval between leading edges of two requests R_i,R_j is not defined. In this case the occurrence of so-called metastable state in the flip-flops of the arbiter is possible.[3,6,7] In addition to the fact that during the metastable state the flip-flop output signals are outside of the tolerance field of logic one and logic zero it should be stressed that occurrence interval of that state is theoretically unlimited. For the security of reliable function of the arbiter the metastable state has to be considered and its unfavorable influence on the outputs A_i must be eliminated.

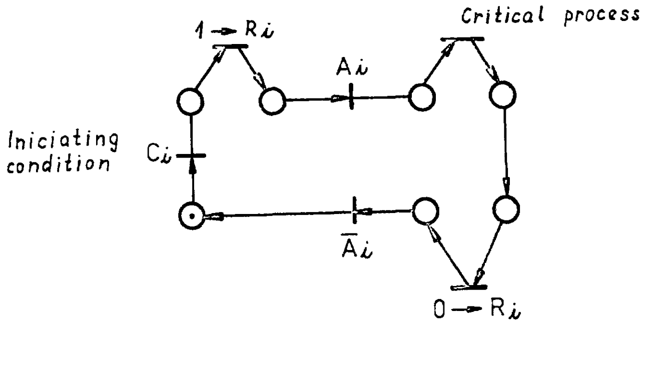


Figure 3

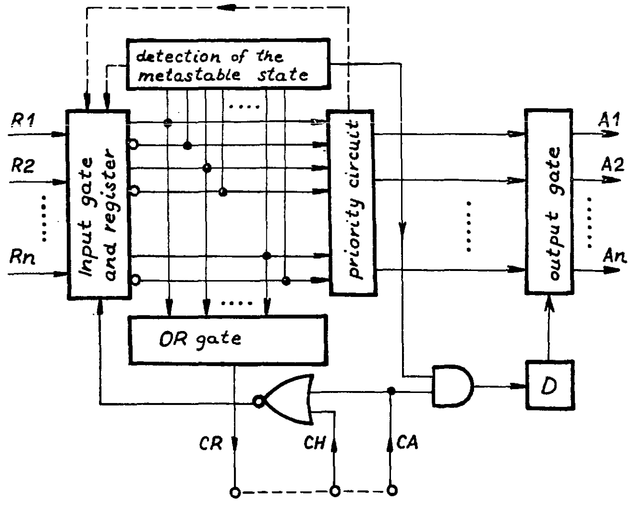


Figure 4

CONNECTION AND FUNCTION

For eliminating the undesirable influence of contingent metastable state the structure of circuit in Figure 4 has been selected. Let us suppose initially that the output CR is connected with the inputs CH,CA and the input signals $R_1 \ldots R_n$ being of zero value. In this case CR=CH=CA=0, the output gate is closed and the input one is open. The block denoted D represents a time delay τ of the leading edge. In this state the contingent requests on the inputs $R_1 \ldots R_n$ are recorded into the input register and at the same time are detected by the OR gate. On coming request R_i the OR gate generates signal CR=1, closing the input gate and in this way recording of further request is inhibited. The priority circuit respecting only the record of the request with highest priority erases in the input register all the contingent lower ones. Provided that during the closing of the input gate arises a metastable state in the input register all the time of its duration a zero output is generated by the detector of the metastable state and through the feed-back forces the relevant bit of the input register in the value one.[3,6,7] This is shortening the time of duration of the metastable state. On finishing of the metastable state the output gate is opened with the delay τ. The connection of the arbiter with four inputs and four outputs is given in detail in Figure 5. By signals M,CR,CH,CA the algorithm of the arbiter can be modified. If M=1, the requests with lower priority in the input register are erased in the described way. If M=0, all the requests which have had logic one before the closing of the input gate continue being recorded. But logic one is transferred only to one of the outputs A_i corresponding to the recorded request with the highest priority. On zero coming to the input of this request the input gate continues—the opposite of the case M=1—to be

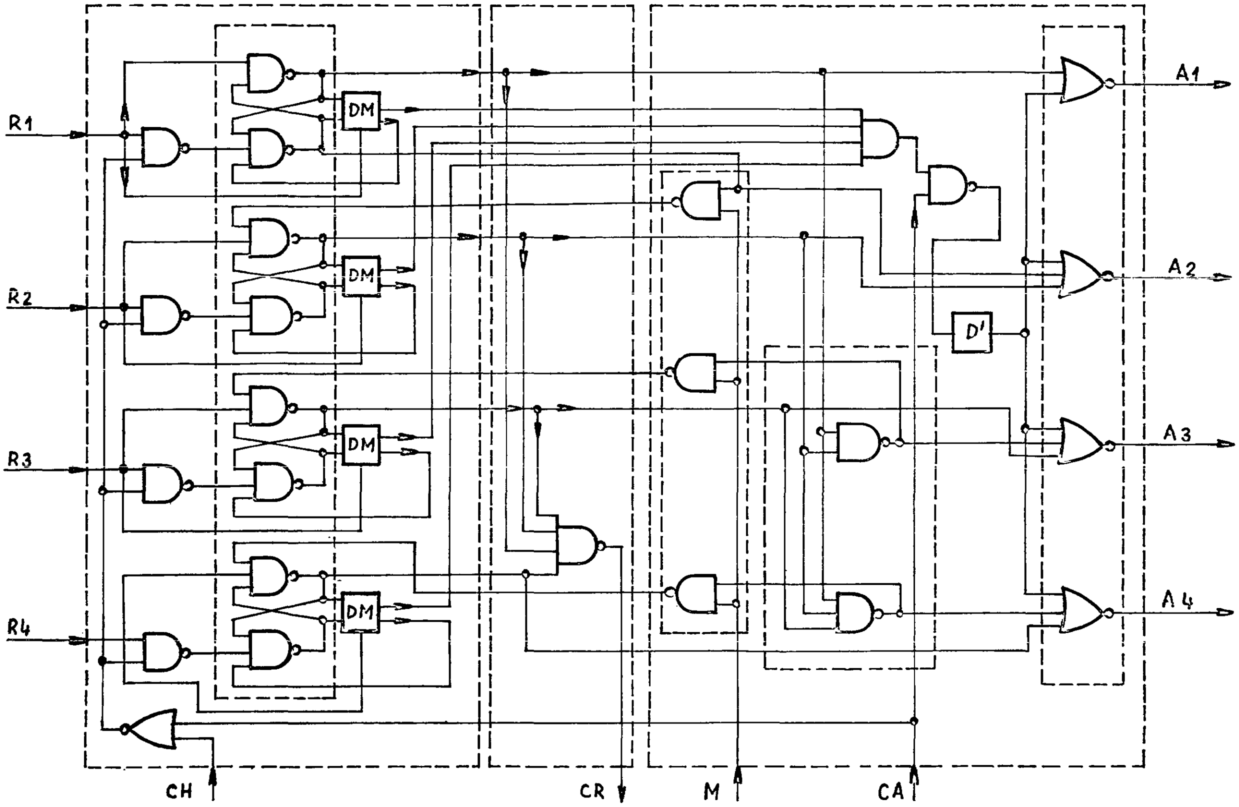


Figure 5

closed and the priority is allocated to one of the remaining recorded requests. This way all the recorded requests are consecutively executed and then on executing the last one, the input register is completely in zero state, signal CR turns to zero and the input gate reopens.

By the general connection of signals CR, CH and CA a further modification of the algorithm is possible. CR and CA represent a couple of speed-independent signals through which the arbiter can be connected with any other speed-independent automata. CR has a purport as a request of the arbiter for the approval of the priority allocation for the input signals $R_1 - R_n$, signal CA with value one makes this allocation possible. The connection of the signal CH either to CR or to CA determines whether the input gate will be closed already after the detection of the first request R_i or only when logic one is appearing on the input CA. Between leading edges of signals CR and CA generally a delay of any length can appear.

The block denoted D' in Figure 5 represents the delay of the trailing edge, blocks denoted DM represent the detectors of metastable state. The metastable state may occur only on appearance of logic one on the input R_i. During this state block DM on the one hand by one of its outputs sets the i-th bit of the input register to the value one and in this way shortens the duration of the metastable state, on the other hand by its second output it inhibits the opening of the output gate.

Signals CR,CA may be used, e.g., to form an arbiter with a larger number of inputs by connecting several smaller arbiters. The principle of such connection is demonstrated in Figure 6. By the concrete connection of signals M and CH of individual "elementary" arbiters a modification of the allocation algorithm according to the needs of desired application is possible.

In Figure 7, we can see Petri-net describing the behavior of the arbiter whose connection corresponds to Figure 5.

Though the connection in the Figure 5 is comparatively simple and contains a low number of gates, there are a number of applications of the arbiter in that the signals CR,CA,CH,M are not used for any specific purpose. In many cases like that also only a two-input arbiter may be used. For these applications the connection given in Figure 8 is more advantageous. An R-S flip-flop represents the input gate, register and priority circuit at the same time. Expanders SN 7460 N are used for the detection of the metastable state.

At the initial state, when signals R_1,R_2 are logic zero the flip-flop is in so-called undefined state when both

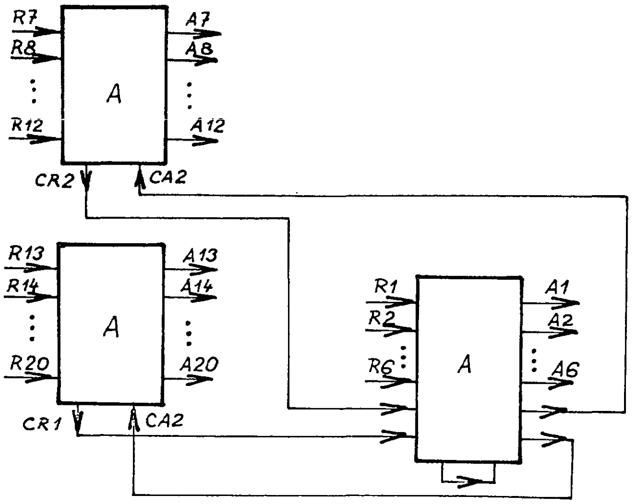


Figure 6

its outputs are logic ones. The voltage on the resistor in this state, the same as in the metastable state, is in the tolerance of logic one and the output gates are closed. In the case of simultaneous arrival of the leading edges of the signals R_1,R_2 the output gates continue to be closed all the time of contingent metastable state and they are opened by the detector of metastable state only when R-S flip-flop turns to the normal state, when the levels of its outputs are in a secure distance to the decision level. At this connection, nevertheless, the detector does not shorten the duration of the metastable state whose length is, theoretically, unlimited.

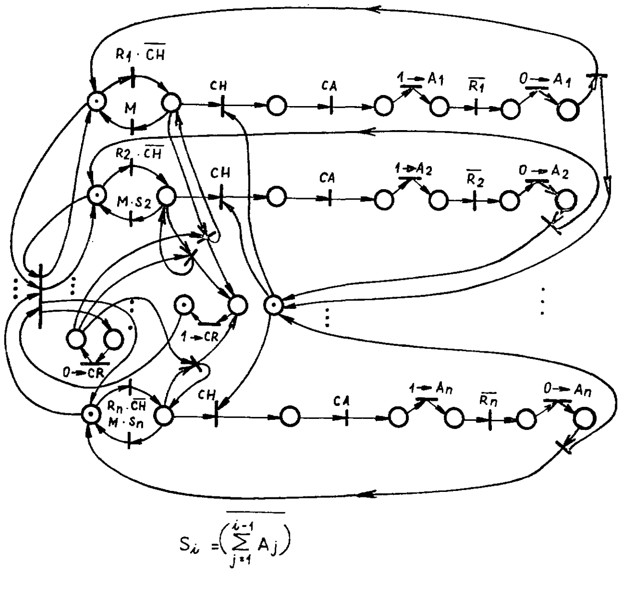


Figure 7

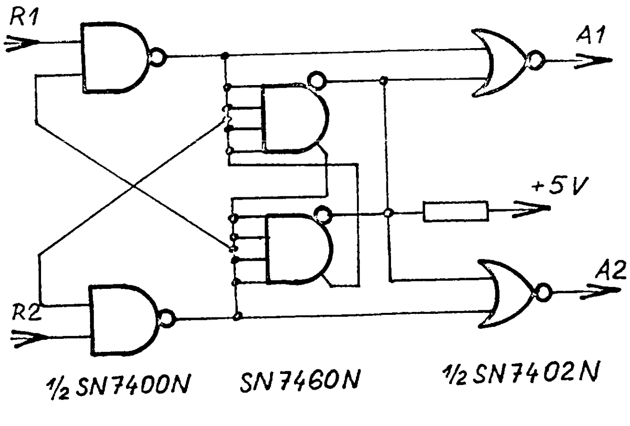


Figure 8

If on the contrary an arbiter with a larger number (e.g., eight) of inputs is needed and the speed requests do not allow use of the cascade connection (Figure 6), it is possible after some modification of the connection to use standard modules MSI but at the cost of lower universality.

THE EXPERIMENTAL RESULTS

For testing the function of the arbiter a sample connected according to Figure 5 has been built. On this sample the reliability of function even for the case of the occurrence of metastable states has been tested. The delay of the trailing edge in the block D had been adjusted $\tau = 30$ nsec. Three kinds of circuit delay have been measured. (See Figure 9.) The delay TLH 1 is defined as the length of the interval between the leading edges R_i and A_i when the others $R_j \neq R_i$ are of logic zero. TLH 2 is defined as the length of the interval between trailing edge R_j and leading edge A_i when $R_i = 1$ and $R_k = 0$, $k \neq i,j$. THL is defined as the length of the interval between trailing edges of signals R_i

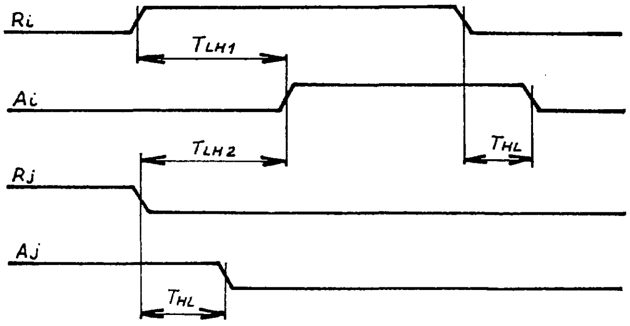


Figure 9

and A_i. The found out values in nsec are given in the following table:

TLH 1	TLH 2	THL		Remark:
90	110	20	i=1, j=2	without metastable state
135	—	—	i=1, j=2	with metastable state
90	110	40	i=2, j=1	without metastable state, regarding the priority of R_1 at metastable state it changes to j=1, j=2.

For the realization of the circuit the integrated circuits of series SN 74 N produced by Texas Instruments have been used. In the case of higher speed requirements the same connection may be realized on faster elements, e.g., series SN 74 S.

EXAMPLES OF APPLICATION

Besides the application of the arbiter as a control module in the systems with the structured hardware,[1,4] the circuit may find a broad application in classic computer and NC systems. In Figure 10, an application of arbiter for synchronization of external signal X by the clock signal Cl is demonstrated. The arbiter with two inputs is making use of the described principle of shortening and limiting the length of duration of the metastable state, flip-flop used is of the type T, sensible to the leading edge of the signal Cl. Applicating the arbiter we avert the occurrence of metastable state in the flip-flop T. The contingent signal X_T of the value one passes namely through the arbiter at least in the time TLH 1 after the leading edge of the signal Cl, e.g., for the mentioned realization of the arbiter in 135 nsec. If the period of the clock Cl is long enough the constant value of the signal X_T in the proximity of the leading edge of the clock is guaranteed.

In Figure 11, we can see a typical application of the arbiter for the control of communication of modules $M_1 \ldots M_n$ with a shared source e.g. with the memory SM and by that as well for the control of the communication on the bus. The module M_i transmits the logic one on the signal R_i as request for the communication with SM. The arbiter reports the allocation of the priority by logic one on the signal A_i. M_i holds the logic one on the signal R_i during the communication with the SM, on the arrival of logic zero on the signal R_i arbiter is released for further operation. The interconnection of signals CR,CA with SM makes the priority allocation possible only in the moment when SM is ready for the communication. The connection of the input M, $M = R_{n-2} + R_{n-1} + R_n$, guarantees to the modules M_{n-2}, M_{n-1}, M_n the allocation of SM after a certain time even in the case of "continuous" queue of requests $R_1 \ldots R_{n-3}$ having in the arbiter higher priorities.

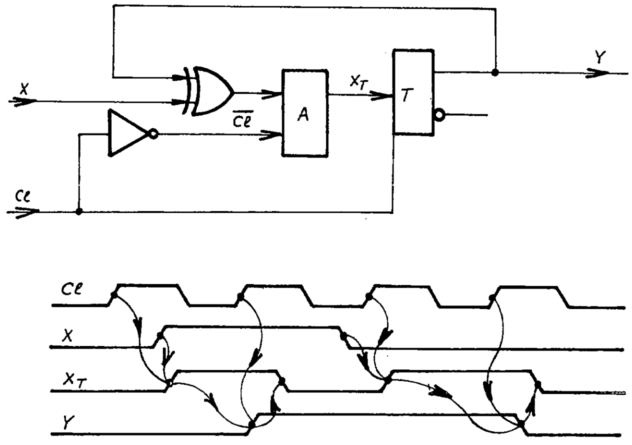
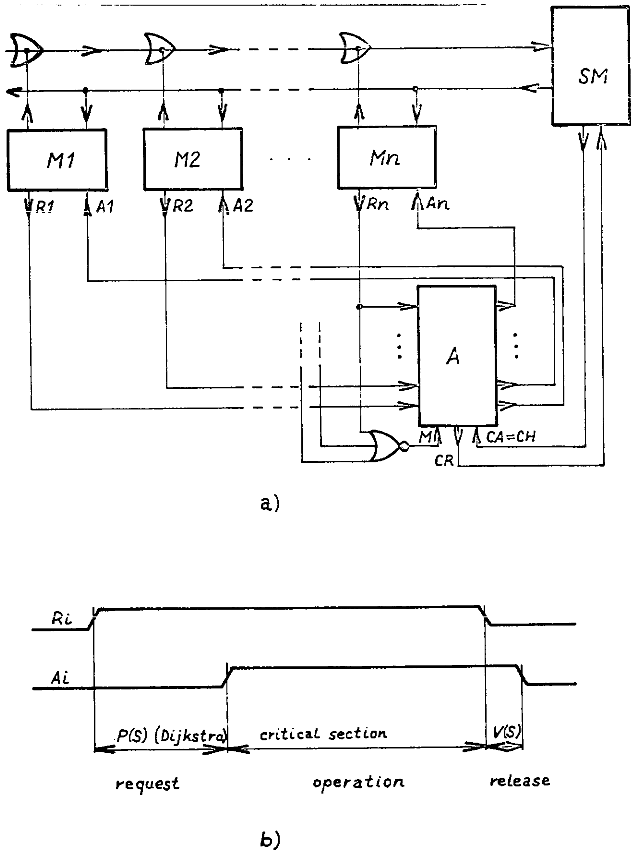


Figure 10



a)

b)

Figure 11

REFERENCES

1. Dennis, J. B., "Modular Asynchronous Control Structures for a High Performance Computer," Conference Record of the Project MAC, *Conference on Concurrent Systems and Parallel Computation*, ACM, New York 1970, pp. 55-80.
2. Plummer, W. W., "Asynchronous Arbiters," *IEEE Transactions on Computers*, Vol. C-21, No. 1, January 1972, pp. 37-42.
3. Chaney, Mollnar, "Anomalous Behaviour of Synchronizer and Arbiter Circuits," *IEEE Transactions on Computers*, April 1973.
4. Dennis, J. B. and S. S. Patil, "Speed Independent Asynchronous Circuits," *Fourth Hawaii International Conference on System Sciences*, January 1971, pp. 55-58.
5. Dijkstra, E. W., "Cooperating Sequential Processes," *Programming Languages*, Academia Press, 1968.
6. Pěchouček, M., *The Connection of Flip-flop Circuit for Processing of Asynchronous signals*, PV 292-75, Patent Registration, ĈSSR.
7. Jura S., M. Pěchouček and H. Sechovsky, *The Asynchronous Arbiter*, PV 7726-75, Patent Registration, ĈSSR.

Log-sum multiplier

by J. P. AGRAWAL and V. U. REDDY
Indian Institute of Technology
Madras, India

ABSTRACT

A scheme for implementing a fast parallel multiplier based on a log-sum method is presented. The scheme for an n-bit by n-bit multiplication uses Q, least integer greater than or equal to $(n^2-n)/4$, 4-bit Carry-Look-Ahead (CLA) adders. The worst case time for multiplying two sign + 12-bit numbers is estimated to be 180 ns when TTL 7400 series I.C.s are used. The scheme can easily be modified to use M-bit (M>4) CLA adders if and when available.

INTRODUCTION

Multiplication of an n-bit number $A(a_n a_{n-1} \ldots a_1)$ by another n-bit number $B(b_n b_{n-1} \ldots b_1)$ involves reduction of n^2 summands $a_i \cdot b_j$, i, j = 1,2, ... n, arranged in (2n−1) columns and n rows, into one 2n-bit row $(P_{2n}\ P_{2n-1} \ldots P_1)$, as shown in Figure 1. The summands are generated in parallel using n^2 AND-gates. A number of schemes have been given in the past[1–4] for implementing fast parallel multipliers which are different basically in the procedure used for the reduction of summands. Dadda's[1] and Wallace's[2] schemes are equally fast and take approximately $\log_{1.5}n$1-bit adding-stages (successive additions) to reduce the summands to two G (least integer $\geq 2n\text{-}2\text{-}\log_{1.5}n$)-bit numbers, which may be added using a G-bit Carry-Look-Ahead (CLA) adder or a ripple adder. Wallace's scheme, however, requires a larger number of adders. The carry-save scheme[4] takes (n−1) 1-bit adding-stages to reduce the summands to two (n−1)-bit numbers. Both Dadda's and the carry-save schemes require (n^2-2n) 1-bit full adders (FA) and n 1-bit half adders (HA).

Habibi and Wintz[4] have shown that the carry-save scheme is slower than Dadda's but cheaper since it makes greater use of 4-bit FAs. However, in both the schemes the 4-bit FAs are connected in such a way that the fast carry generation property of the 4-bit FA I.C. chip is not utilized.

The purpose of this paper is to present a log-sum configuration for the reduction of summands, in which the numbers of 1-bit FAs and 1-bit HAs required are the same as in Dadda's and the carry-save schemes. But these adders can be replaced fully by Q 4-bit FAs taking full advantage of the fast carry generation. Q is the smallest integer greater than or equal to $(n^2-n)/4$. Hence, the log-sum scheme is more economical than any other parallel multiplier and at the same time yields a high speed of operation. Another feature of the scheme is that it can easily be modified to adopt longer FAs if and when I.C. technology makes them available.

In the second section, we give a general scheme for reduction of summands using 1-bit FAs and HAs and then modify it for 4-bit CLA adders. In the third section, we present cost and speed analysis for the log-sum multiplier and compare it in the fourth section with other schemes. In the fifth section, we discuss how the scheme can be extended for M-bit FAs, and to include 2's complement multiplications.

THE LOG-SUM MULTIPLIER

In the log-sum method[5] the addition of n rows is performed in $\log_2 n$ adding-stages where each adding-stage reduces the number of rows by a factor of 2. Therefore, the adder-rows (row of adders to add two rows) required for the first, second, ... $(\log_2 n)$th adding-stages are $\frac{n}{2}, \frac{n}{4}, \ldots$ 1, respectively. This method of addition is used to yield economic and fast multiplication.

Preliminary scheme

The n rows, shown in Figure 1, are grouped into (n+k)/4(=L) sets (k=0,1,2 or 3 such that L is an integer), the last set consisting of (4−k) rows and the others 4 rows each. The 4 rows in a set are reduced to two rows by first stage addition and then reduced to yield one (n+4)-bit row by second stage addition, as shown in Figure 2 for n=7. As illustrated in the Figure, (n−1)-bit and (n+2)-bit adder-rows are required for first stage and second stage additions, respectively. Also, it is seen that in this configuration (i) the outputs of two successive sets always overlap

$$\begin{array}{ccccccccccl}
 & & & & & a_n & \cdots & a_3 & a_2 & a_1 & \\
 & & & & & b_n & \cdots & b_3 & b_2 & b_1 & \\
\cline{5-11}
 & & & & & a_n \cdot b_1 & \cdots & a_3 \cdot b_1 & a_2 \cdot b_1 & a_1 \cdot b_1 & \text{row 1} \\
 & & & & a_n \cdot b_2 & \cdot & \cdots & a_2 \cdot b_2 & a_1 \cdot b_2 & & \text{row 2} \\
 & & & \cdot & & & & \cdot & & & \\
 & & \cdot & & & & \cdot & & & & \\
 & & a_n \cdot b_{n-1} & a_{n-1} \cdot b_{n-1} & \cdots & a_2 \cdot b_{n-1} & a_1 \cdot b_{n-1} & & & & \text{row } n-1 \\
 & a_n \cdot b_n & a_{n-1} \cdot b_n & a_{n-2} \cdot b_n & \cdots & a_1 b_n & & & & & \text{row } n \\
\hline
P_{2n} & P_{2n-1} & P_{2n-2} & P_{2n-3} & \cdots & P_n & \cdots\cdots & P_3 & P_2 & P_1 & \\
\hline
\end{array}$$

Figure 1—Multiplication of two n-Bit Numbers A $(a_n \cdots a_1)$ & B $(b_n \cdots b_1)$

in n-bit locations and (ii) each mth set has an half adder in (n+4m−3)th column. Therefore, outputs of the mth and (m+1)th sets are added using one n-bit adder-row, feeding the overflow bit from this to the corresponding half adder in (m+1)th set. This corresponds to the third stage addition. In the same manner, only n-bit adder-rows are required for each of the 4th, 5th, . . . and ($\log_2 n$)th adding-stages. The total number of 1-bit full/half adders required in this scheme may, therefore, be written as,

$$N=\frac{n}{2}(n-1)+\frac{n}{4}(n+2)+\left(\frac{n}{8}+\frac{n}{16}+\ldots+\frac{n}{2^j}\right)n$$

$$=\frac{3n^2}{4}+\frac{n^2}{4}(1-2^{-j})=n^2-n, \text{ where } j=\log_2(n/4);$$

which is same as required by Dadda's scheme.

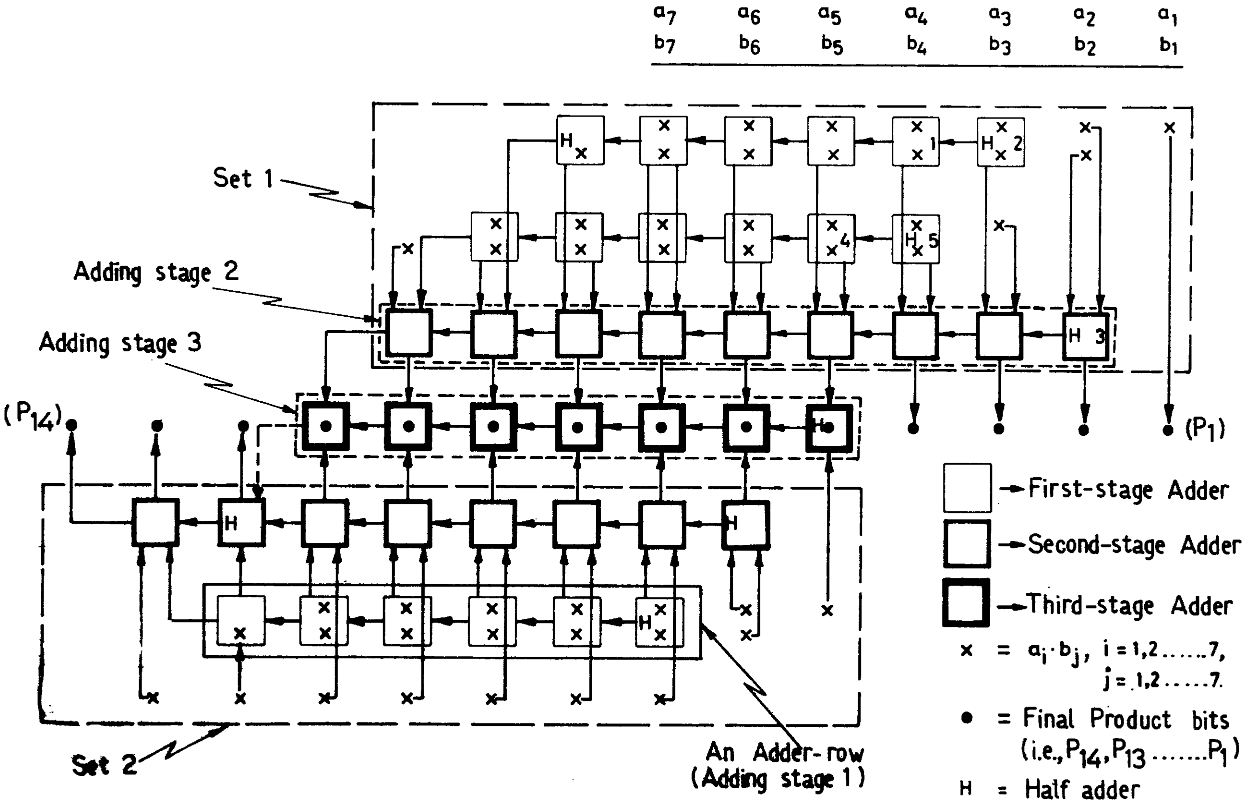


Figure 2—General scheme for 7-bit x 7-bit log-sum multiplier

Final scheme

A fast and economic multiplier is realized by rearranging a few adders in the above described scheme as follows.

(i) All the adders in the final propagation path which includes the final adding-stage are aligned in one row.

(ii) After (i) has been completed some adders are moved from one adder-row to the other and rearranged such that the number of adders in each row (except in one row if N/4 is not an integer) is a multiple of 4. An adder that is moved from one adder-row is rearranged in another adder-row, as far as possible, corresponding to the equivalent or higher adding-stage. This is illustrated in Figure 2 for n=7, where five adders marked 1 to 5 are rearranged to yield the final scheme.

(iii) The strings (or rows) of 1-bit full/half adders in (ii) are replaced by the strings of 4-bit adders.

(iv) In general, the least significant bit (lsb) adder in each adder-row is a half-adder. Such an HA in a particular adding-stage is interchanged with a corresponding FA in a lower adding-stage if it allows to reduce carry-dependence in that adder-row. This is illustrated in Figure 4 which gives final scheme for a 12-bit x 12-bit multiplier.

If the above four steps are carried out in the order listed above, $\log_2 n$ successive additions (i.e., one 4-bit addition in each adding-stage) would yield approximately $\frac{n}{2}+1$ lsb's ($a_1 \cdot b_1$ being the first lsb) of the final product, as seen from Figures 3 and 4. This is so because the two inputs to the final-adding stage are the sum of first $\frac{n}{2}$ rows and the sum of last $\frac{n}{2}$ rows, which are displaced by $\frac{n}{2}$ bits.

COST AND SPEED ANALYSIS

In this section we give expressions for the cost and the multiplication time of the log-sum multiplier. Costs and speeds of the 7-bit x 7-bit and 12-bit x 12-bit multipliers are computed from these expressions assuming Texas Instruments (TI) TTL 7400 series I.C. chips. Prices have been taken from the 1974 "Cramer Electronics Catalogue" and delays from 1974 Texas Instruments 'The TTL Data Book for Design Engineers'.

Cost estimation

The log-sum multiplier requires n^2 AND-gates for generation of summands and Q 4-bit FAs for reduction

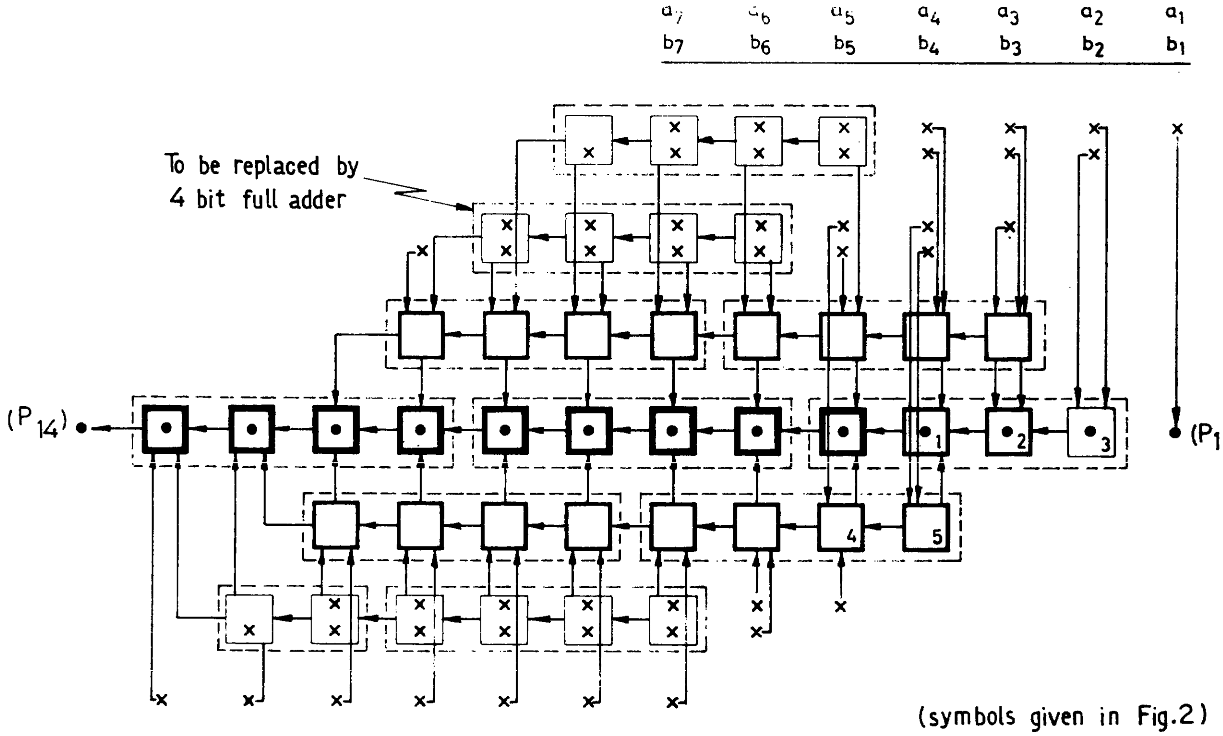


Figure 3—Final scheme for 7-bit x 7-bit log-sum multiplier

of summands. Hence, the cost of the log-sum multiplier may be written as

$$C = c_1 \cdot R + c_2 \cdot Q$$

where c_1 and c_2 are the prices of a quad 2-input AND-gate chip and a 4-bit FA chip, respectively. R is the least integer greater than or equal to $n^2/4$ and Q is the least integer greater than or equal to $(n^2-n)/4$.

The 7-bit x 7-bit log-sum multiplier requires 49 AND-gates and, hence, 13 quad 2-input AND-gate chips (SN 7408), and, 11 4-bit FAs (SN 7483A). The 12-bit x 12-bit multiplier requires 36 quad 2-input AND-gate chips and 33 4-bit FAs. The corresponding costs are about $40 and $115, respectively.

Speed estimation

As indicated earlier, $\frac{n}{2}+1$ lsb's of the final product are obtained in $\log_2 n$ 4-bit successive additions. The time required to obtain remaining product bits is the carry-propagation delay from $\left(\frac{n}{2}+1\right)$th bit to $(2n-1)$th bit. Therefore, the multiplication time required by the log-sum multiplier is given as

$$T_M = t_g + t_{ad} \cdot \log_2 n + t_c \cdot \left(2n - 2 - \frac{n}{2}\right)/4$$
$$= t_g + t_{ad} \cdot \log_2 n + (3n-4) \cdot t_c/8$$

where t_g is the time required for the generation of summands (i.e., propagation delay of one 2-input AND-gate), t_{ad} is the addition time for a 4-bit FA and t_c is the carry-propagation delay of the 4-bit FA.

Maximum values of t_g, t_{ad} and t_c, as given in TI 1974 catalogue, are 27 ns, 24 ns and 16 ns, respectively. Hence, the worst case multiplication time for $n=7$ and $n=12$ cases are 130 ns and 180 ns, respectively.

A 7-bit x 7-bit multiplier was built on a printed circuit board, using the scheme given in Figure 3, for use in a special purpose signal processor.[6] The multiplication time required for each possible combination of inputs was measured. The maximum time was found to be less than 110 ns which is well below the estimated worst case value of T_M.

COMPARISON WITH OTHER SCHEMES

Cost comparison

The log-sum multiplication scheme is most economical of all the array multiplier schemes, since it employs minimum number of 4-bit FAs. Each of the other schemes requires 1-bit and/or 2-bit FAs in addition to 4-bit FAs or can be implemented using larger number of 4-bit FAs. The adders required by the carry-save scheme, which is more economic than any other past schemes, are compared with those required by log-sum

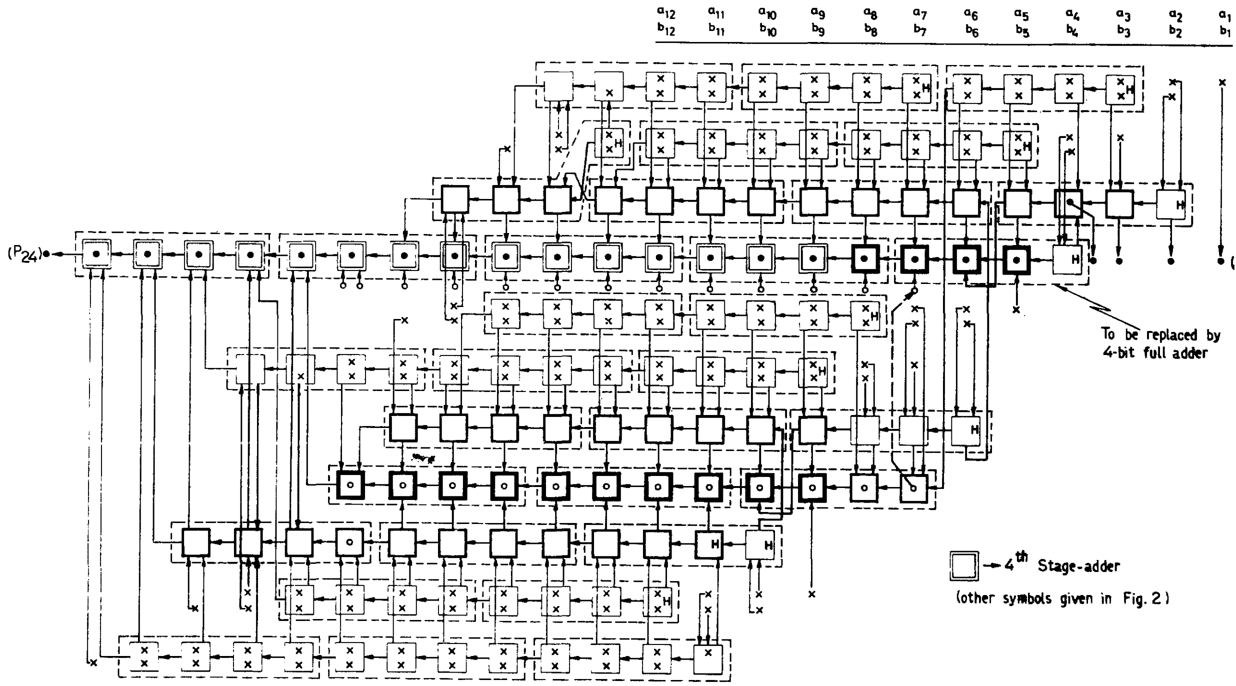


Figure 4—Final scheme for 12-bit x 12-bit log-sum multiplier

TABLE I

		Number of adders required		
k	Scheme	4-bit FAs	2-bit FAs	1-bit FAs/HAs
0	Log-sum	Q	—	—
	Carry-save	$Q-\frac{3n}{4}$	n	n
1	Log-sum	Q−1	1	—
	Carry-save	$Q-\frac{2n+2}{4}$	n	—
2	Log-sum	Q−1	—	—
	Carry-save	$Q-\frac{n+2}{4}$	—	n
3	Log-sum	Q	—	—
	Carry-save	Q	—	—

scheme for various values of k (defined in an earlier section) in Table I.

As seen from this table, the cost of the log-sum multiplier is at most equal to that of carry-save multiplier.

Speed comparison

Dadda's scheme requires roughly $\log_{1.5}n = 1.71 \log_2 n$ adding stages. The multiplication time required by Dadda's scheme, therefore, is approximately given by

$$T_{MD} = t_g + t_{ad} \cdot 1.71 \log_2 n + \frac{2n-2-1.71 \log_2 n}{4} \cdot t_c$$

assuming 4-bit FAs throughout (including the final addition). It is clear from the expressions for T_M and T_{MD} that for all n, $T_{MD} \geq T_M$. The worst case values of T_M and T_{MD} for n=7, 12 and 32 are given in Table II. If a G (let G be the smallest integer≥)-bit CLA adder, built from Schottky TTL I.C. chips, is employed to add the final two numbers in Dadda's scheme, T_{MD} reduces significantly. However (i) use of Schottky TTL I.C.s increases the multiplier cost enormously, and (ii) T_{MD} reduces to less than or equal to T_M only for large values of n(>32).

Pezaris[7] has given a 40-ns 17-bit x 17-bit array multiplier, using basically the carry-save scheme. However, he has used ECL logic and, hence, the cost and speed of his multiplier cannot be compared with those of log-sum multiplier using TTL logic, presented in this paper.

TABLE II

n	Log-sum scheme T_M(ns)	Dadda's scheme T_{MD}(ns)
7	130	170
12	180	230
32	330	450

DISCUSSION

The configuration presented in this paper yields a fast multiplier using the optimum number of 4-bit CLA adders. The scheme can be adapted to M-bit CLA adders by making the number of adders in each row a multiple of M instead of a multiple of 4 in step (ii) of the second section of this paper.

The scheme has been presented for the multiplication of two sign+n-bit numbers. The sign of the product is generated separately. Baugh and Wooley[8] have given an algorithm for 2's complement array multiplier, in which the multiplication process is same as given in Figure 1 except that the number of rows increases from n to n+2. Therefore, the 2's complement multiplication can easily be implemented by log-sum scheme.

A 4-bit x 4-bit multiplier from TI (requiring two chips, SN 74284 and SN 74285) costs approximately $26, and, the worst case and typical values of multiplication time are 60 ns and 40 ns, respectively. The same multiplier based on log-sum scheme costs approximated $11, while the worst case and typical values of multiplication time are approximately same as above.

Finally, the log-sum scheme fully exploits both the cost effectiveness and speed effectiveness of the 4-bit FAs and, therefore, is well suited for multipliers in special purpose hardware requiring high speeds of operation.

REFERENCES

1. Dadda, L., "Some schemes for parallel multipliers," *Alta Frequenza*, Vol. 34, pp. 349-356, March 1965.
2. Wallace, C. S., "A suggestion for a fast multiplier," *IEEE Trans. Electronic Computers*, Vol. EC-13, pp. 14-17., Feb. 1964.
3. Braun, E. L., *Digital Computer Design.* New York, Academic Press, 1963.
4. Habibi, A. and P. A. Wintz, "Fast multipliers," *IEEE Trans. Computers* (Short notes), Vol. C-19, pp. 153-157, Feb. 1970.
5. Kuck, D. J., "Illiac IV software and application programming," *IEEE Trans. Computers*, Vol. 17, pp. 758-770, August 1968.
6. Agrawal, J. P., V. U. Reddy and H. Renganathan, *Design and Test Evaluation of Arithmetic Unit for FFT Processor*, Technical Report, CSD/TR/A12-2/19, Sept. 1974, IIT, Madras.
7. Pezaris, S. D., "A 40-ns 17-bit by 17-bit array multiplier," *IEEE Trans. Computers* (Short Notes), C-20, pp. 442-447, April 1971.
8. Baugh, C. R. and B. A. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Trans. Computers*, Vol. C-22, pp. 1045-1047, December 1973.

Distributed information systems

by GRAYCE M. BOOTH
Honeywell Information Systems
Phoenix, Arizona

ABSTRACT

Distributed information systems represent an increasingly important trend to computer users. Distributed processing is a technique for implementing a single logical set of processing functions across a number of physical devices, so that each performs some part of the total processing required. Distributed processing is often accompanied by the formation of a distributed database. A distributed database exists when the data elements stored at multiple locations are interrelated, or if a process (program execution) at one location requires access to data stored at another location. Examples of how these techniques are being used are provided, with comments on the advantages and disadvantages of the distribution of processing and databases in the current state-of-the-art.

INTRODUCTION

This paper discusses distributed information systems—what they are, and why they will become increasingly important to computer users. Distributed systems—also called distributed database systems—are in the early stages of development as yet, so much of this discussion will be theoretical. Whenever possible, however, practical examples will be supplied from systems now being planned, implemented, or in use.

DEFINITIONS

Information network

A distributed database system always exists within an information network environment. An information network is a combination of information processing facilities, data communications facilities, and endpoint facilities. Together these support the movement and processing of files, programs, data, messages, and transactions. An example of a complex information network is shown in Figure 1.

Data communications facilities include the transmission ("telephone") lines, coupling devices such as modems, and concentration devices such as multiplexors and terminal concentrators. The endpoint facilities include terminal devices, and also satellite processors used both as endpoints and for localized information processing.

Distributed processing

Distributed processing exists when a single logical set of processing functions is implemented across a number of physical devices, so that each performs some part of the total processing required. Two types of distributed processing have been defined, horizontal and hierarchical (or vertical).

In horizontal distribution all devices cooperate at an equal level, logically, to perform a set of tasks. There is no hierarchical relationship among the devices.

In vertical distribution the interconnected devices form a hierarchy, sharing tasks in a structured way with each component to some degree controlled by the higher-level member(s) of the hierarchy.

In a complex system, such as the Figure 1 example, both types of distributed processing may be present. Horizontal distribution is shown in the example be-

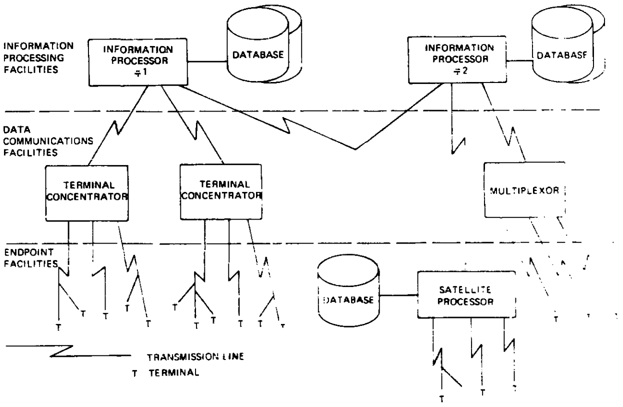


Figure 1—Information network

tween the two information processors, and hierarchical distribution is shown between Information Processor #2 and its satellite processor.

Distributed database

Whenever multiple processing devices are configured in an information network the possibility of a distributed database exists. In the broadest sense the data stored at all locations could be considered to form a distributed database. However, for practical purposes a distributed database exists only when the data elements at multiple locations are interrelated, or if a process (program execution) at one location requires access to data stored at another location.

A distributed database may consist of a single copy of a set of information, divided into increments which reside at multiple locations. This form is called a partitioned database.

A distributed database may also consist of a set of information, all or selected parts of which is copied at two or more locations. This form is called a replicated database. A single distributed system may make use of both of these forms of database.

EXAMPLE SYSTEMS

This section presents a small sample of distributed database systems which are now in the planning and/or implementation process.

Example #1

The first example, shown in Figure 2, is a horizontally distributed system. Two large-scale information processors, located remotely from each other, are linked using communications facilities.

The two computers communicate by exchanging files. A file may be a source-level program; in the company shown most program development takes place in City A, and copies of new programs are transmitted to City B for use there.

A file may also represent a job (an executable program and its accompanying input/output data), which is transmitted from one information processor to the other for load leveling purposes.

This example represents a "loosely-coupled" distributed system. The two information processors are largely independent, and exchange data only occasionally. A distributed database is not required to support this mode of operation.

Example #2

A manufacturing control system (Figure 3) is the second example. In this system the large-scale information processor is not directly involved in real-time process control. It maintains the master database, which is used for overall scheduling and control of the manufacturing process.

At the next level in the hierarchy there are several satellite processors, implemented using minicomputers. Each satellite handles a part of the manufacturing process, and maintains its local database, which is a subset of the central database and contains only the data applicable to the local task. This is an example of a replicated database.

The next lower level of the hierarchy consists of terminal concentrators. These minicomputers link the satellites to the lowest-level devices, concentrating data from many devices onto a few communications links and/or direct cables to each satellite processor.

The lowest-level minis (in some cases microcomputers are used) monitor and control the factory equip-

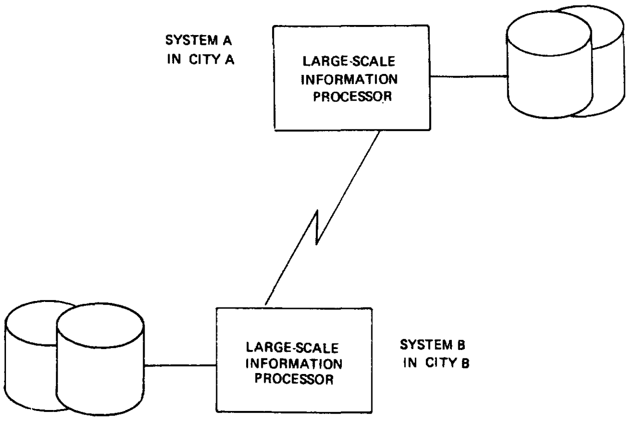


Figure 2—Horizontally distributed system

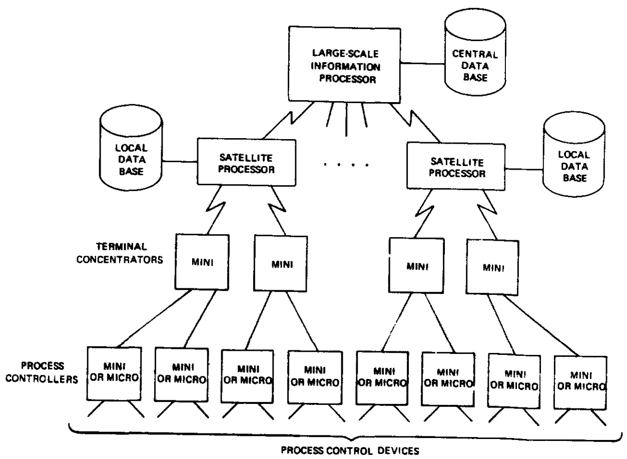


Figure 3—Hierarchically distributed system

ment. Process status information is sent up to the satellite processors, while control commands are received from the satellites to guide the manufacturing process.

Distributed system architecture

The two major forms of distributed system architecture are horizontal distribution and vertical (or hierarchical) distribution. In a very complex system both forms may be used.

HORIZONTALLY DISTRIBUTED PROCESSING

The horizontal distribution of processing functions involves the interconnection of two or more components which are logically equal. Note the term "logically"; the components may be physically unlike and of different capacity and power. The important aspect is their logical relationship within the distributed system.

In a horizontally distributed system multiple information processors most often cooperate to exchange jobs and/or transactions so that the total workload is suitably distributed. An example is shown in Figure 4.

The example shows three information processors, geographically distributed but interconnected via data communications facilities.

In a configuration of this type each information processor will normally handle jobs/transactions which originate locally. The interconnections can be used for load leveling among the three information processors. Candidates for load leveling could range all the way from compilations to complex jobs which require that the referenced files be transferred between cities with the jobs.

This illustrates one trend among current users; the interconnection of hitherto independent computers to gain the advantages of load leveling and common access to all files (which may form a distributed database).

HIERARCHICALLY DISTRIBUTED PROCESSING

Probably the most basic rule in designing hierarchically distributed systems is to spread the processing load up and down the hierarchy by locating functions where they can be performed with the best cost/performance ratio. Functions which are required repeatedly and with quick response are moved out (or down) into the hierarchy as far as feasible. Other functions less often executed and/or with less stringent response requirements tend to move upward toward the center of the hierarchy.

As functions are relocated within the hierarchy the data which supports these functions must move with them, leading to the formation of distributed databases.

An example three-level hierarchy is shown in Figure 5. At the top of the hierarchy is a central computing complex, consisting of one or more large-scale information processors. At the next level there are several satellite processors, placed in factory locations, warehouses, or bank branches, depending on the type of application. At the lowest level there is a potentially large number of controllers, to which the terminals and/or other input/output devices (such as process-control equipment) are attached.

It is easy to visualize hierarchical systems with fewer levels or with more levels, although three is probably most typical today. A two-level hierarchy might omit the terminal controllers and handle the terminal/device interfaces directly from the satellite processors. A four-level hierarchy might separate each satellite processor into a local information processor and a terminal concentrator.

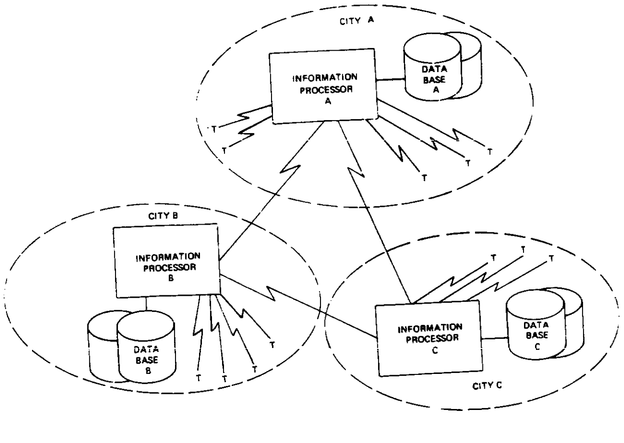


Figure 4—Interconnected computer service centers

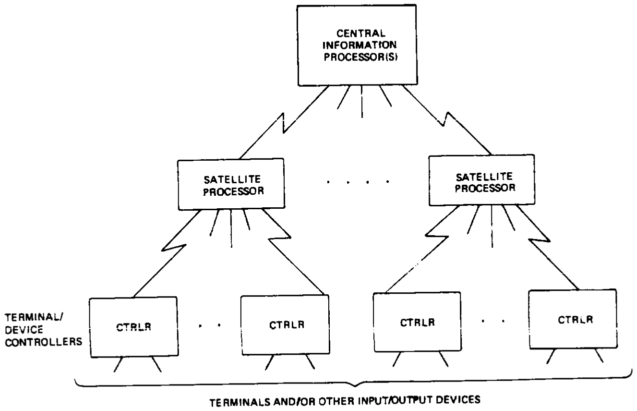

CENTRAL INFORMATION PROCESSOR(S)
SATELLITE PROCESSOR
. . . .
SATELLITE PROCESSOR
TERMINAL/ DEVICE CONTROLLERS
CTRLR
. .
CTRLR
CTRLR
. .
CTRLR
TERMINALS AND/OR OTHER INPUT/OUTPUT DEVICES

Figure 5—Computer hierarchy

Distributed database architecture

In many distributed processing systems there is a strong probability that a distributed database will be required. Distributed databases can be separated into two categories:

- Partitioned databases
- Replicated databases

PARTITIONED DATABASES

The first category of distributed database exists when a conceptual database is separated into sections and spread across multiple computers. The term "conceptual" is used because in general a single database is not created and then partitioned; instead, the database is designed as a logical entity but actually only created in the form of its partitions. The separate sections, because of their interrelationship, logically form a single database.

Database access partitioning

Database partitioning often follows the natural distribution of database access requirements.

As an example, consider a wholesale company with two information processors. Their total database will probably be split into two partitions, and one attached to each computer. This separates both the processing load and the database accesses between the two information processors. If the company operates nationwide within the United States, the information processors might be located as shown in Figure 6.

In the example system it is logical to locate database partitions heavily accessed by east coast users at the eastern information processor and locate those most accessed by west coast users at the western information processor. Accesses from intermediate locations must be equitably distributed between the two computers.

An important reason for geographical grouping of data is the cost of transmitting data to/from remote locations. In general, the shorter the transmission distance the lower the cost.

In the Figure 6 example provision must be made for at least some use from outside of the area where the database partition is located. The east coast database partition will no doubt sometimes be accessed from west coast locations, and vice versa. In a distributed database system such accesses must be allowed. However, the great majority of the accesses to each section of the database will originate locally. If this is not the case, the database has not been partitioned correctly.

Vertical hierarchy partitioning

Another example of the use of a partitioned database is shown in Figure 7. In this case the database is partitioned over a hierarchy, rather than across a horizontally distributed system.

This example shows a processing hierarchy which corresponds to the organizational hierarchy—a fairly typical case. The large-scale information processor handles corporate-level processing, with a corporate database attached. Each division has its own satellite processor—two of these are shown. Each divisional satellite processor has associated with it a satellite database. This example assumes that there is no duplication of information among the databases shown. In effect, therefore, the data elements at all locations form a corporate database, which has been partitioned across the hierarchy.

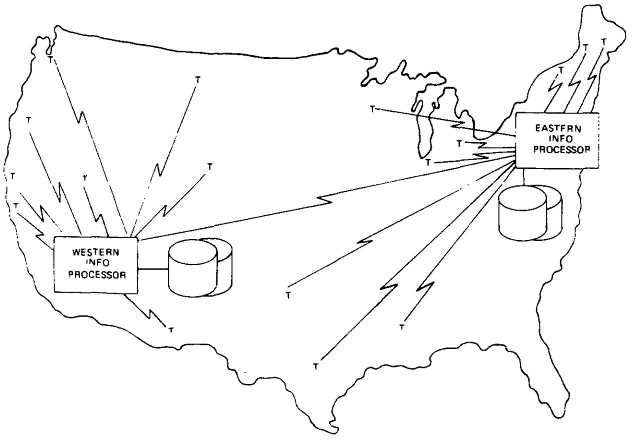


Figure 6—Geographically partitioned database

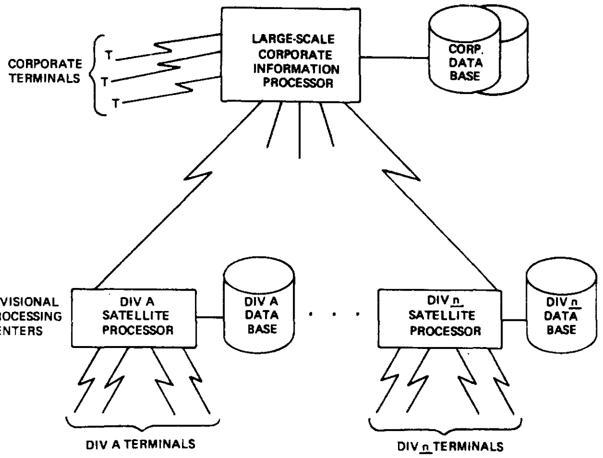

CORPORATE TERMINALS
T
T
T
LARGE-SCALE CORPORATE INFORMATION PROCESSOR
CORP. DATA BASE
DIVISIONAL PROCESSING CENTERS
DIV A SATELLITE PROCESSOR
DIV A DATA BASE
. . .
DIV n SATELLITE PROCESSOR
DIV n DATA BASE
DIV A TERMINALS
DIV n TERMINALS

Figure 7—Vertically partitioned database

Information is exchanged in both directions across the hierarchy. Corporate-level reports require information from the divisional database for consolidation with corporate data. Data also travels down the hierarchy, as goals and budgets are established at the corporate level and are passed down to the divisional level.

REPLICATED DATABASES

The replication of all or part of a database at two or more locations is another way to create a distributed database.

An example of the use of a replicated database is shown in the Figure 8 illustration of a complex banking system formed of the interconnection of two hierarchies.

In this system the central information processors maintain the master database of customer accounts, with the total set of customer records partitioned between the two computers. Each satellite processor maintains a database containing the accounts for its local customers. Each local database is created by copying the necessary information from one of the partitions of the central database.

The local databases are essentially work files, refreshed each night from the master database. On-line activity during the day is posted to the local databases; these are used for withdrawal authorization and similar functions. At night all activity is batched and used to update the master database, from which new local databases are then created, and the cycle repeats.

Distributed database system use

The two forms of distributed system architecture—horizontal distribution and hierarchical distribution—and the two categories of distributed database—replicated database and partitioned database—can be combined in a variety of ways.

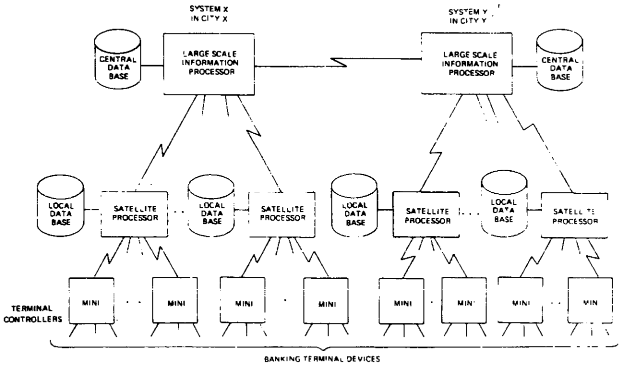


Figure 8—Complex banking system

DYNAMIC LOAD LEVELING

A distributed system can be used to dynamically spread the total processing/database access load across the available computers. In general, this mode of use is associated with horizontally distributed systems.

In a relatively straightforward situation an organization might have three identical information processors, as shown earlier in the Figure 4 example.

If all three information processors have the same capabilities, then jobs/transactions can be moved freely among them. It may be desirable to execute a specific job in a particular computer because it requires access to a database associated with that computer. On other occasions it may be desirable to move jobs, possibly accompanied by their data, from an overloaded information processor to one which has available capacity. The choices for load leveling can be summarized as follows:

- Move the process to the data
- Move the data to the process

In a batch-oriented environment the tendency is to transmit a job, or a subdivision of a job, to the information processor where the database is located. Because a batch job often does extensive processing against the database, and database size is typically large, it is more economical to transmit the job and its input/output data than to transmit the database.

In the Figure 4 example, if a job is entered at City A but requires access to Database B, then the job will normally be transmitted to Information Processor B, executed there, and output returned to City A if necessary.

In inquiry/response applications, in contrast to batch processing, usually only a small part of the database is accessed. It is therefore feasible to consider transmitting the data to the process and returning only updates (if any) to the computer where the data is stored. In this case the process, as specified by its process-state and program, may be considerably larger than the data involved.

STATIC LOAD LEVELING

In the second method of organizing a distributed database system the total load is leveled by preassigning functions and database segments to processors statically.

This method is most often used in a hierarchically distributed system, which may be formed of unlike computers; i.e., large-scale information processors, minicomputers, terminal/device controllers, and microcomputers.

In this situation control of a distributed database is somewhat less complex than in the preceding case. The functions which must be supplied are: (1) provide remote as well as local access to parts of the distributed database; and (2) move parts of the database up and down the hierarchy.

Referring to the complex banking system shown earlier in Figure 8, in this system enough information is included in each satellite processor's database to handle approximately 80 percent of the transactions which originate at the local terminals. Deposits and withdrawals can be handled by the satellite, provided that the customer's home branch is within the satellite processor's local area.

Conditions which the satellite cannot handle include withdrawals for a customer whose account is in a remote branch, and complex transactions such as loan applications and credit card applications. For this particular bank 80 percent of all transactions can be handled by application programs resident in the satellite processors, working with their local databases. The complex transactions are sent up to the large-scale information processors; these have a more complete set of application programs and a much larger database.

Perhaps the most complicated case to handle occurs when a customer enters a branch remote from the branch at which his account is maintained, and requests a withdrawal. Approximately 3 percent to 5 percent of all transactions are of this type.

In this case the withdrawal transaction is passed on by the satellite processor where entered, and travels up the hierarchy to the nearest central information processor. From there it is passed, via the other information processor if necessary, to the satellite which handles the customer's home branch. The withdrawal is processed by that satellite, using its local database, and the response is returned to the satellite where the transaction entered.

The banking system is a good illustration of the principles of distributing functions—application processing—and database information as close as feasible to the point of transaction origination. It also illustrates the principles of centralizing the handling of small numbers of complex transactions which cannot economically be handled by each of the satellite processors.

SUMMARY

Today the computer industry and its users are in the early stages of a developing trend toward the use of distributed database systems. Not all information system users will follow this trend; many will continue to be served satisfactorily by more conventional centralized system structures.

However, advances in both hardware and software technology will make distributed architectures attractive to an increasing number of users. Information networks will become more and more common in the next five to ten years. Accompanying these information networks and distributed system architectures will be an increased use of distributed databases.

These distributed database systems, although complex, will provide a degree of efficiency and cost/effectiveness often impossible to achieve in centralized system architecture.

ACKNOWLEDGEMENTS

I am indebted to many of my colleagues at Honeywell Information Systems, especially the Distributed Processing Technology Group under M. Canepa, for their significant contributions to this work.

Error detection in data base systems*

by MICHAEL HAMMER
M.I.T. Laboratory for Computer Science
Cambridge, Massachusetts

ABSTRACT

Incorrect data poses a serious impediment to the effective use of computerized data bases. Conventional approaches to the design and implementation of automated data error detection systems are inadequate for large and complex data bases. Partly, this derives from the inherent intricacy of the problem, with decisions being required as to what checks to perform, how and when to do the checking, and how to respond should an error be found; writing procedures to accomplish these functions is a difficult programming task. Also at fault is the unrealistic and overly simplistic view of data correctness embodied in most contemporary systems.

"Intelligent" data checking systems are required, which possess more extensive knowledge of the data base environment. They will need to understand the structure of the world which the data base models; the way the data base is used, and the relative importance of its various components; the sources of the errors that might occur and the costs of detecting them; and the patterns and rates of errors that actually do occur. Such a system would then be in a position to detect a wide range of errors, allocating its resources in a systematic fashion and responding appropriately to different error situations.

INTRODUCTION

It is a truism that any decision-making system is only as good as the data which it uses; this is especially true of computer-based information systems. Yet while the number and complexity of such information systems have been dramatically increasing in the last few years, supported by great enhancements in the technology of data base management systems, comparatively little has been done to assure the quality and reliability of the data on which these systems depend. Many computerized data bases in use today suffer from high error rates in the data they receive, and are consequently riddled with bad data. And with incorrect data, the most efficient and sophisticated system is well-nigh useless.

There are numerous potential sources of error in a computer data base. Some of these originate in the computer system itself, deriving from such causes as hardware failure or interfering concurrent users; however, these are generally secondary sources and can be controlled by existing technology. The major error problem lies in data that arrives at the computer already corrupted. The possible reasons for this corruption are manifold, including human error in the initial recording or formatting of the data; faulty transcription of the data, either by human or machine (e.g., OCR scanners); accidentally lost, forgotten, or delayed data; deliberately falsified or omitted data; and the like. It is also useful to include in our notion of erroneous data those cases where no error has occured in the transcription or transmission of the data, but rather where the data faithfully represent an illegal or impossible event. That is, the error is not in the data, but in the information it conveys.

In this paper, we consider the problem of detecting errors in data bases. We begin with a re-examination of the concept of data errors, and consider both the problems that they can cause and the difficulties that can arise from attempts to detect them. This leads us to consider the issues and problems of error detection, and the inadequacies of the current technology in this field. We proceed on the basis of this analysis to propose an alternative way of thinking about data integrity, an approach which has major implications for the design of error detection systems.

DATA ERRORS AND DATA INTEGRITY

It is worthwhile to re-examine the concept of error in the context of a data base. A data base is not just a collection of values; it serves as a model of some limited universe. At every point in time, the contents of the data base represent some configuration of that application world. Every such world is governed by some set of rules that determines the legitimacy or plausibility of its states, that specifies which configurations of

* This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under contract number N00014-75-0661.

the world are reasonable. Consequently, every correct version of the data base must obey these rules as well. If the values in the data base do not represent a legal configuration of the world, then there is an error in the data base.

In addition, every world has rules which do not constrain its instantaneous versions, but rather restrict the legitimate transitions between its valid configurations. Such rules are not violated by particular instances of the data base, by illegal transformations of the data base from one consistent version to another. For example, the rule that "salaries do not decrease" would be violated by a change to the data base which decreased some salary, though both the before and after states of the change might be internally consistent and legitimate.

The foregoing concept has been referred to as the "logical consistency" or "semantic integrity" of a data base.[1,3,4,5,7] This property is violated when the data base no longer reflects a legal world instance, or when a change is made to the data base that does not obey the constraints on world actions.

It is useful to consider two generalizations of the foregoing idea. First of all, many application domains are not governed exclusively by rules of a black-or-white character, which specify certain situations as perfectly legal and all others as totally illegitimate. Rather many worlds make extensive use of the notions of probability and likelihood. For example, it may not be impossible for an employee to get a 100% raise, but it is implausible. Therefore, we could not immediately classify data attesting to such a condition as being erroneous, but we would have to look at it askance; and should we note many such instances, our skepticism would begin to rise.

The second generalization is that errors are really just one kind of exceptional condition. Data that reflects situations that are in the norm can, in some sense, be said to have been anticipated, and so their occurrence requires no system response. However, unusual data, that either is faulty or represents a surprising turn of events, is noteworthy and demands special attention and response. Thus, any violation of the rules of a world model, whether caused by faulty data or by an illegal activity, requires detection.

We can identify several types of errors in a data base, which are of increasing complexity and difficulty to detect.

1. Most simply, an individual value which is inappropriate for the field which holds it. For example, the value may be of the wrong data type (e.g., a salary which is not numeric) or of the wrong magnitude (e.g., a salary less than $4000).[6]

2. An inconsistency between different fields of the same record. For example, an individual's salary may not be consistent with his job classification. Here we encounter the problem of localizing the error: though we know which record is faulty, we have to determine which of the two fields is the one causing the problem. We may have to draw upon additional information to resolve this.

3. An inconsistency between field(s) of one record and fields(s) of related record(s). For example, there might be a rule that no individual's salary may be greater than that of his manager. Should this rule be violated by the data base at some point, the error could only be localized to one of two possible offending records.

4. Some global pattern out of order in some set of records or in the data base as a whole. This would be a violation of some restriction, not on individual entities of the application domain, but on collections of them. These global patterns may involve aggregate functions (the average salary of all employees should not exceed $12,000), functionality relationships (every department has exactly one manager), and subset requirements (every manager is also an employee). These global pattern violations usually indicate not the presence of some particular faulty value, but rather of a general trend which violates expected norms. In these cases, it is often not the data that is wrong, but the events that they report, or sometimes even the the rules which define right and wrong in this environment.

5. Missing data. This notion includes blank fields, obsolete values (the new data is missing), entire records which are referenced but cannot be found, and lacking data which cannot be localized.

DATA ERRORS VERSUS DATA CHECKING

The hazards of allowing erroneous data into a data base hardly need belaboring. They include the obvious results of improperly executed operational activities and faulty decisions based on incorrect information. But bad data can also cause the malfunction of application programs that use the data base and can even degrade the performance of the data management software itself, sometimes to the point of failure; this occurs because programs frequently implicitly assume that the data which they manipulate are semantically reasonable and consequently satisfy various criteria. On a more systemic level, erroneous data can destroy the confidence of an organization's personnel in the entire information system, with a deleterious impact on morale; frequently the result is an atmosphere which fosters even higher error rates, due to negligence and disinterest.

While it is easy to recognize the dangers of allowing erroneous data into a data base, there are also potential hazards in attempting to prevent this eventuality from occurring. The major pitfall in this regard is an overzealous commitment to the notion of data correctness, whereby the error detection process becomes an end in itself. This condition results from a loss of perspective, wherein the actual dangers of the errors that do occur are not accurately assessed nor the costs of their detection weighed against the damage they can cause.

In reality, not all data errors are of equal significance in the context of a particular information system. Different parts of a data base are used for different purposes, and consequently have unequal degrees of importance; this ought to be recognized by the error detection system. Any aspect of the data base may be said to have both a "sensitivity" and an "impact" with respect to its being in error. For example, the salary field in a personnel file is very sensitive to error; any faulty value will give rise to an improper action (namely, the cutting of an incorrect salary check). On the other hand, the age field, if used primarily in projecting future manpower requirements, is comparatively insensitive; it would take many erroneous values to diminish the utility of the aggregate of the age field values. Yet the potential impact of the latter error situation may be much greater; a bad high level decision, in contrast to one inaccurate action. It would be appropriate, then, for a data checking system to expend its resources in proportion to the importance and severity of the errors it is trying to detect.

If the foregoing idea is not appreciated, a disproportionate amount of system resource may be invested in error checking, well beyond what it realistically deserves. This investment may manifest itself in the expense of constructing a powerful and "complete" error checker, or in the actual processor time allocated to this function. There may be more insidious side effects as well: over-emphasis on error detection may impair the performance of the data management system as a whole. Excessive timidity in the presence of possible errors can delay the entire decision-making process, or may even reduce the over-all functionality of the whole information system. On the other hand, an unwarranted over-confidence in a faultily performing error checking system can lead to a false level of confidence in a data base that may be in fact contaminated with much bad data.

CONVENTIONAL APPROACHES

The state-of-the-art in automatic error detection/correction has not been highly developed. Most data editing that is currently done is usually confined to the simplest validity checks, assuring that various fields are of the appropriate data types or are within specified ranges. The structure of most error systems is that they are comprised of a collection of hand-coded, special purpose procedures (written in assembly code or some conventional programming language), each of which is designed to monitor some particular transaction with the data base, and assure both that the transaction is legal and that it leaves the data base in a consistent state. These programs are generally entirely ad hoc, and follow no well-defined approach or methodology.[2]

This approach to error detection does not satisfactorily address the problem of complexity. Any data base that represents a real world system of interest is almost certain to be governed by a large and rich set of complex rules which delimit the legality of its data values. To reliably and effectively capture these rules in a disconnected set of procedures written in a conventional programming language is a task that strains the effective capability of contemporary programming technology.

In consequence, current data checking systems are frequently very expensive to build and, once built, are often highly unreliable. Furthermore, they are almost impossible to modify in an orderly and consistent way in order to meet new requirements, either in terms of evolving definitions of the errors to be detected or in modified characteristics of the operating environment.

Conventional methods of error detection usually check for error conditions on the occasion of every (primitive) update (insert, modify, delete) to the data base. This is expensive and can also lead to anomalous results. Consider for instance the restriction that no employee is to make more than his manager, and a series of transactions that increments everyone's salary by 8 percent. If an employee's raise is processed before his manager's, application of the test would compare a new salary with an old one and might raise a spurious flag, reflecting a transient condition in the data base.

In addition, current error systems are cumbersome, relying on primitive brute-force techniques to do the data checking; in general, they operate in a very inefficient manner. Partly this is a consequence of the complexity problem; the incorporation of any but the simplest techniques in the system structure described above might topple the entire structure. In another sense, this clumsiness is a result of the rigid and inflexible view of data correctness alluded to earlier. In this view, every datum is viewed as being either correct or incorrect, and all errors are assumed to be catastrophes of equal magnitude. There is no appreciation of the fact that while some fields must be error-free, others (because of the way they are used in the information system) can tolerate some measure of error; nor does it take into account the concept that the severity of an error might be measured in terms of its deviance from a legitimate value. Thus there is usually little relation between the importance of an error and the amount of resource expended to detect it.

Nor does the conventional system structure provide a systematic capacity for response to an error situation. Once the presence of an error has been detected, it is then necessary to localize the error and specify exactly which data value is at fault; in many cases, the state-of-the-art is incapable of doing this satisfactorily. After an error has been identified, there are several possible responses. The options include rejecting the erroneous data; allowing it into the data base, signalling its occurrence and/or marking it as questionable; putting it into a suspended status, pending the receipt of additional clarifying information; attempting automatic error correction. The choice of response should be determined by the importance of the error, the reliability of the error detection procedure, and the particulars of what causes the error. However, current systems are inflexible in this regard. They are especially weak in automatic error correction, relying on ad hoc techniques which can easily go astray.

All of these problems are greatly exacerbated in the context of large data bases with high data input rates. There the requirements of the data checking system can easily overwhelm the available system resources. These circumstances have resulted in data bases that are full of bad data despite extensive data checking, and whose utility is grossly restricted because of the unreliability of much of the data.

In some applications, the current technology is adequate, because of the simplicity of the world of application and the consequent limited range of possible data errors. However, the current state-of-the-art will not be able to meet the demands of data bases of increasing size, complexity, and sensitivity. As data bases are used to model ever larger and richer domains, and are relied on for more critical decisions, the need for reliable data will be ever more pressing and more difficult to satisfy.

AN ALTERNATIVE APPROACH

In view of the problems that have arisen as a result of a non-systematic approach to error detection, we propose the following basic principles to be adhered to in the design of error detection systems.

1. Expect errors, and be prepared to fail because of them in a "soft" manner. Do not design the kernel of the information system under the hypothesis that all data is correct, and then delegate to an error-checking subsystem the responsibility of bringing about that state of affairs. The goal of perfectly correct data is an unattainable one. We can minimize the presence of certain errors subject to various constraints, but cannot expect to eliminate them altogether.

2. Formulate the rules of the application world first; then relate them to the constructs of the data base; and then identify potential logical errors in terms of the data base transactions. Attempt to ensure the completeness and consistency of the world model before proceeding further.

3. Evaluate the importance of the various error types with respect to the decision process, considering both sensitivity and impact. Allocate the resources of the data checking system in line with this evaluation.

4. Anticipate the kinds of errors that may occur, identifying their sources, estimating their relative frequencies, and determining the costs of defecting them. Be prepared to adapt to changes in any of these factors: in the definitions of what constitutes an error; in patterns of actual error occurrence; in relative importance of errors; and in the costs of error detection. Learn from historical situations and detect emerging patterns as well.

We believe it is possible to construct an error-detection system in accordance with these principles, and are in fact currently engaged in such an effort. Below, we summarize the most salient features of the system design.

1. The system will be declarative, rather than procedural. That is, the rules governing the legitimacy of data values will not be implicitly embedded in the code of the procedures that perform data-checking, but will be stated as explicit assertions by an individual knowledgeable in the domain which the data base is modelling. The primitives used in the expression of these rules (called constraints) will be chosen to be naturally descriptive of the kinds of worlds most frequently represented in data base systems; thus a non-programmer would be able to describe the constraints directly, without programmer intervention.[1,3,4,5,7]

The declarative approach has numerous advantages over procedural techniques, besides ease of expression. Because the rules are all listed in a single place, it is possible to make modifications to them in a reliable and simple fashion. Since the procedure that uses the constraints to check the data is only written once, it is possible to verify its accuracy reliably and hence have confidence in its correct operation. In addition, the system (rather than an applications programmer) will bear the responsibility for determining and checking all the appropriate rules for a given transaction. Finally, it will be possible to examine the explicit constraints themselves, looking for incompleteness, redundancy, and inconsistency.

2. The constraints will do far more than express legal ranges of values for specific individual fields; they will describe rich structural aspects of the world in question, thus enabling far more sophisticated error checking than is currently done. In order to detect many kinds of errors of this sort, it is necessary for the checking program to possess a good deal of rich and complex "knowledge" of the domain in question. Conventional data base management facilities, and data editing programs that rely on them, do not have the capability to represent and utilize knowledge of the requisite form.

3. The occasions at which errors will be detected will correspond to "conceptual transactions" or "structured operations" with the data base[3,5]; these are appropriately selected sequences of data base transactions that form representations of real-world events. It is these complex transformations that actually occur in the world that the data base models, and so it is appropriate to check them (rather than each of their primitive constituents) for validity. This approach will

avoid many spurious situations and will facilitate suitable response to the detection of an error.

4. The system will be designed to operate independently of the actual data base system, and to reside on a remote processor. This processor will be dedicated to the task of data checking, but it will be able to communicate with the data base facility itself by means of a communication link. It will utilize the constraints which model the domain of the data base to examine data to be submitted to the data base; should it detect an error, the offending transaction can be identified and appropriately handled.

In order to rule on the validity of a transaction, the checking processor may need to inspect some of the values in the data base itself; it will obtain the items it needs by issuing a request for them to the data management system over the communication channel, receiving the results in the same way. As time goes on, the checking processor will build up a local version of those parts of the data base which it most frequently requests, thus obviating the need for much of the data transmission.

This strategy opens up several important possibilities. First of all, by detecting errors closer to the source of their occurrence, the opportunity for their timely correction is greatly enhanced; thus the overall performance of the information system is improved. Furthermore, the data checker can be made relatively independent of the actual data management system which is responsible for the data. Variations in the "back-end" data manager need only have a minor reflection in the operation of the "front-end" error detection processor. This is especially important in an environment where one conceptual data base may be distributed among several data management systems. A final potentiality is for distributed data checking, with multiple front-ends, each processing a separate stream of source data. This can address the problem of checking the correctness of large data bases with high input data rates.

5. The system will adopt a flexible view of the concept of error. While perfectly correct data is a laudable goal, in reality it is an unattainable one, and compromises must be effected in its pursuit. The system will make the necessary trade-offs, in the realm of efficiency versus accuracy, in a systematic fashion. Rather than swamping its resources with a futile and inappropriate effort to detect all errors, the system will concentrate on validating those items that require the highest levels of accuracy. Resources will be allocated to checking other fields based on their relative importance. The data base administrator will provide the system with the information regarding the comparative significance of various data items in the decision making process.

In deciding how to allocate its resources, the system will also consider the effectiveness of the various checks. It will have to determine the actual frequencies of various errors; combining this information with an estimate of the cost of detecting each of the errors in question and with a measure of the importance of the item being validated, the system will be able to proceed in an optimal fashion. The system will also be adaptive, automatically adjusting to changing patterns of errors.

6. The system will possess the degree of "world-knowledge" needed to process errors in an intelligent fashion. The first aspect of this is error localization, identifying exactly which data values are at fault. Once an error has been detected and localized, an appropriate response must be made. The range of such responses is great, and the one to choose may depend on numerous factors and, for a given error type, may even change over time. The system will be flexible enough to accommodate many possible responses, and will be in a position to select the right one.

The most ambitious of all responses is, of course, the automatic correction of the error in question, to restore the original value. The system will possess a facility for the data base administrator to provide a model of the error-making process in the system in question, which describes the loci where, and means by which, errors do occur. In those circumstances which the data base administrator identifies, this model can be utilized to reverse errors that do occur.

The ultimate goal of any error system should be error prevention: the identification of the sources of the errors that actually occur, and the instigation of remedies to forestall their further occurrence. The system will utilize the model of the error process, together with observed patterns of error, to pinpoint the origins of the most common errors and identify them to the data base administrator.

In order to achieve the capabilities just outlined, it will be necessary to utilize novel implementation techniques of a statistical character, that will enable the system to attain maximum data reliability subject to the constraints imposed by available resources. For example, the system could sample the incoming data in order to determine the actual rates of different transaction types and errors. If some errors are found to occur extremely infrequently, then those parts of the system used for their detection can be temporarily disabled. This may allow some erroneous values to slip by undetected, but this should be acceptable if the fields in question can tolerate the expected level of error. The sampling should be reactivated from time to time to ensure the constancy of the error rate. This approach results in application of resources where they are most needed. Other useful statistical information would include various summaries of the data base contents (in order to detect emerging global patterns) as well as patterns of error types and erroneous fields. These can be used to pinpoint the sources of the most common errors as well as indicate when an error is not in the transaction but in the master file itself.

To satisfactorily address the manifold demands of large and complex data bases, it will be necessary to rely on heuristic techniques that gain efficiency at the expense of perfect accuracy. For example, we might associate with various fields of the data base different levels of reliability, based on observed error patterns. These measures can help in localizing an error indicated by a data base conflict. Another idea would be to utilize concepts from "fuzzy logic," and abandon the notion of situations being only correct or incorrect. We might evaluate some predicate at a given time and find it to be "very true", so entirely satisfied that in all likelihood it would take a very long time for a normal series of transactions to make it turn false; we could then reasonably suspend the further checking of this predicate for a long time.

ERROR PREVENTION

Perhaps more important than these techniques which can be used to implement or facilitate error detection, are those measures that should be taken to prevent errors from occurring in the first place, or diminish their potential for harm if they do occur. Primary (and most obvious) of these is the adequate training of those who collect, record, and transmit data. But these individuals must also be provided with incentives to encourage accuracy in the submission or processing of data; someone who feels he derives no benefit from an information system is unlikely to expend much effort on its behalf. There are other "human factors" issues that can have enormous impact on the error rate. For example, the design of the forms on which data is recorded is a very important consideration, which when improperly done, can have disastrous consequences. Such questions as the layout of a form, the size and location of the different fields it contains, the redundancy of the information it carries, and the extent to which it is prefilled, all require careful thought.

The global structure of the information system will also impact the reliability of the data base. For example, it is important to provide multiple levels of protection, so that should an error get past one level of detection it will not immediately get into the data base and linger there forever. One mechanism to achieve this is a feedback system, whereby parts of the data base are directed to those responsible for (or familiar with) them, and then subjected to human review. Distributed data checking is another technique that can contribute to data reliability. It results in more timely error detection and more rapid correction of errors because of proximity of the detection to the source. It also makes for greater system efficiency, with the data base computer only checking for errors that cannot be caught at the source.

The structure of the files that actually comprise the data base also has an effect on the reliability of the data base. Some contemporary information systems are based on a single large file, consisting of a great number of identically structured records, each with many fields. Such a structure does not facilitate the prevention or detection of errors. Every transaction has potential access to all fields of a record and, should it go awry, can have unlimited consequences within the record. Thus, to prevent disastrous situations, all transactions require inspection and validation. A different kind of file structure can be more conducive to data correctness. For example, a multi-file structure wherein critical data is not stored together with less critical information enables us to concentrate our checking resources on transactions that manipulate the former. Similarly, a separation of stable, seldom changing data from that which is highly volatile can prevent erroneous transactions with the latter from influencing the former. Indeed, it may be appropriate to impose a multi-level structure on the data, for example by age. The first level might be the most recent data, not yet fully verified as to its accuracy; the second, the currently active data, into which new data is moved as it is validated; and the third, old seldom-used data, including statistical summaries. The record structure should also be carefully chosen to encourage intra-record checking, and to facilitate inter-record checking when it must be done.

CONCLUSION

In this paper, we have presented the framework of an approach to the construction of "intelligent" error detection systems. Further research is needed to formulate and develop the principles which underlie the advanced approach to error processing described above; having developed these concepts, it will be necessary to assure their viability and practicality by building systems that embody them, and use these systems in realistic problem environments. There are several major conceptual areas which require substantial development in the course of this research: the representational scheme for data base constraints; optimization techniques for efficient data checking; communication between a front-end processor and the back-end data base; measures of criticality and reliability of data, and the heuristics that make use of them; error correction as driven by a model of the error process. Nevertheless, this approach holds great promise for attaining the needed levels of reliability in the large and complex data bases of the future.

REFERENCES

1. Florentin, J. J., "Consistency Auditing of Data Bases," *The Computer Journal* 17(1), February, 1974, pp. 52-58.
2. Gosden, J. A., "Large Scale Data Base Systems—Current

Deficiencies and User Requirements," in D. Jardine (ed.), *Data Base Management Systems*, North-Holland Publishing Company, Amsterdam, 1974, pp. 105-115.

3. Eswaran, K. P. and D. D. Chamberlin, "Functional Specifications of a Subsystem for Database Integrity," *Proceedings of International Conference on Very Large Data Bases*, Framingham MA, 22-24 September 1975.
4. Graves, R. W., "Integrity Control in a Relational Data Description Language," *Proceedings of ACM Pacific Conference*, San Francisco CA, 17-18 April 1975.
5. Hammer, M. M. and D. J. McLeod, "Semantic Integrity in a Relational Data Base System," *Proceedings of International Conference on Very Large Data Bases*, Framingham MA, 22-24 September 1975.
6. McLeod, D. J., "High Level Domain Definition in a Relational Data Base System," *Proceedings of 1976 ACM SIGMOD-SIGPLAN Conference on Data*, Salt Lake City, UT, 22-24 March 1976.
7. Stonebraker, M. R., *High Level Integrity Assurance in Relational Data Base Management Systems*, Electronics Research Laboratory Report ERL-M473, University of California, Berkeley CA, 16 August 1974.

A framework for federal health data collection

by N. PHILLIP ROSS
Department of Health, Education, and Welfare
Rockville, Maryland

and

MEYER KATZPER
Systems and Informations Analysis
Rockville, Maryland

ABSTRACT

Large scale health data collection under Federal auspices is examined in this paper. The system planning for establishing the data base is intertwined with the vagaries and complexities of the Federal health administrative structure. This paper addresses the problem of how data can be structured in such a manner that it will give rise to maximum utility across the Federal bureaucracy. At present, minimum linkage between Federal agency data bases is possible. Establishment of linkages between different types of data collected is vital. In the Development of the data base design, careful considerations is given to the human element within the system structure. Mechanisms for human interface in the monitoring and control of data collection and processing are vital if the data is to be put to use.

The new opportunities that can be presented by large scale data collecting are explored briefly in this paper. The necessity for common coding systems and decisions as to storage of the data and form of recording are addressed. The data base is considered as the core element in an entire system design concept, which cuts across Federal, State and local levels.

INTRODUCTION

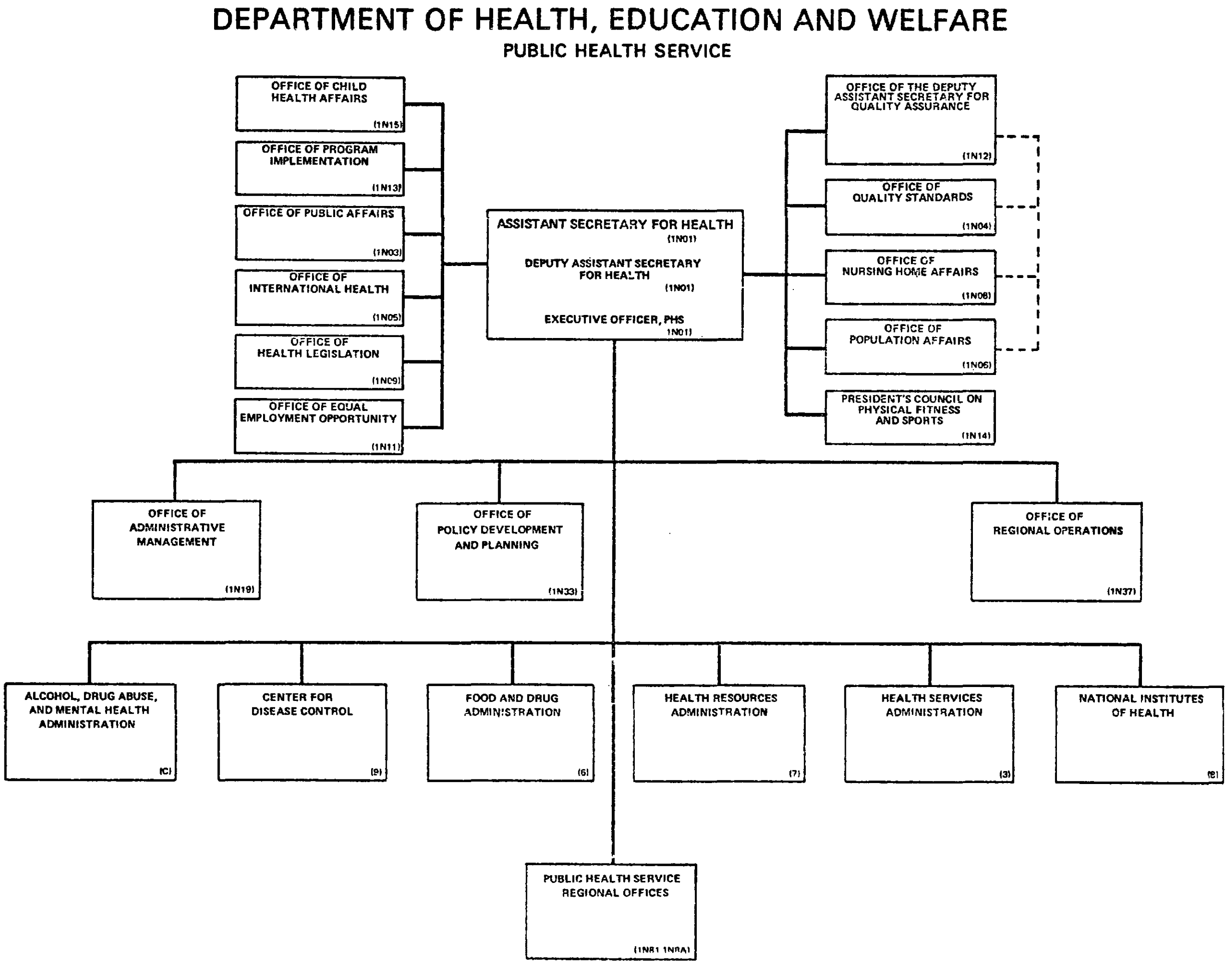
The Federal collection of health related data is continually expanding. This large scale data collection is intimately linked to the goals of many health programs. In most general terms the motivation behind these programs is to improve health care and lower costs. The amount of data collection and analysis necessary for the accomplishment of these goals is made feasible by computer technology. This paper deals with some aspects of the requirements of data systems to achieve programmatic goals. In particular, we indicate the framework and some of the planning and procedures for the data collected to be effectively transferred and used throughout the Federal health administration structure. Data bases that are set up can be used to achieve objectives under the jurisdiction of different departments only if they are well designed and easily accessible to all users. For this purpose appropriate technical considerations are insufficient. In terms of the entire system, there must be responsible people as checkpoints to review, pass on and act on the data collected. Such a simple question as the length and mode of storage of the large volume of data collected requires a full understanding of the possible uses of the data in the system. In view of the costs of the administrative structure and the burden of collection, structuring, processing and disseminating the data, a plan must be formulated to optimize possible data usage.

In this paper, we deal with the problems of a political and organizational nature within the health bureaucracy only from the point of view of the flow of data. However, this alone requires a framework for coordination and synthesis of activities and a network of responsible people to monitor and review data usage.

DATA BASE DEVELOPMENT

With the ever increasing involvement of the Federal government in the financing and regulation of the nation's health care, enormous fragmented bureaucracies, with numerous departments looking after specific aspects of health care, are being created. Under the Public Health Service alone, there are seven (7) separate agencies (See Table I) with different and overlapping responsibilities in areas of national health care. This expansion of the Federal bureaucracy has not been accompanied by the development of an organizational structure capable of coordinating the various components and, as a result, data collection activities are duplicated and health data systems are generated

TABLE I


DEPARTMENT OF HEALTH, EDUCATION AND WELFARE
PUBLIC HEALTH SERVICE
OFFICE OF CHILD HEALTH AFFAIRS (1N15)
OFFICE OF PROGRAM IMPLEMENTATION (1N13)
OFFICE OF PUBLIC AFFAIRS (1N03)
OFFICE OF INTERNATIONAL HEALTH (1N05)
OFFICE OF HEALTH LEGISLATION (1NC9)
OFFICE OF EQUAL EMPLOYMENT OPPORTUNITY (1N11)
ASSISTANT SECRETARY FOR HEALTH (1N01)
DEPUTY ASSISTANT SECRETARY FOR HEALTH (1N01)
EXECUTIVE OFFICER, PHS 1N01)
OFFICE OF THE DEPUTY ASSISTANT SECRETARY FOR QUALITY ASSURANCE (1N12)
OFFICE OF QUALITY STANDARDS (1N04)
OFFICE OF NURSING HOME AFFAIRS (1N08)
OFFICE OF POPULATION AFFAIRS (1N06)
PRESIDENT'S COUNCIL ON PHYSICAL FITNESS AND SPORTS (1N14)
OFFICE OF ADMINISTRATIVE MANAGEMENT (1N19)
OFFICE OF POLICY DEVELOPMENT AND PLANNING (1N33)
OFFICE OF REGIONAL OPERATIONS (1N37)
ALCOHOL, DRUG ABUSE, AND MENTAL HEALTH ADMINISTRATION (C)
CENTER FOR DISEASE CONTROL (9)
FOOD AND DRUG ADMINISTRATION (6)
HEALTH RESOURCES ADMINISTRATION (7)
HEALTH SERVICES ADMINISTRATION (3)
NATIONAL INSTITUTES OF HEALTH (8)
PUBLIC HEALTH SERVICE REGIONAL OFFICES (1NR1 1NRA)

with overlapping purposes. At the State and local level this duplication results in the imposition on local health agencies of an intolerable amount of redundant data collection and routing requirements. For example, Table II lists several data elements wh'ch Missouri State hospitals are required to collect for the different Federal, State and local (FSL) agencies listed in Columns 2 through 8. In almost all instances, the same data element is required by more than one agency and on a different form. Because of this duplication and the resulting burden on providers of health data, the health data users of Missouri are presently examining the possibility of entering into a Cooperative Health Statistics System* (CHSS) in which the users will define their needs, establish a "data broker" who will collect and aggregate data from all providers within the State and provide each user with needed data.

In order for the data collected within any specific region, such as in the Missouri region, to be useful on a national scale, an overall system must be designed which relates Federal, State and local data needs both in terms of basic sets of core data and the need for special data collection specific to a given area or activity. The entire system will only be of use if it is designed in such a way that necessary information can be retrieved from one source and linked with information from other sources. Data within the system must be in unit record form and the units must be of sufficient disaggregation to permit the study and analysis of variables affecting the supply and demand of health resources. Not all variables of interest can be maintained in a single base. Linkage between data bases will allow the integration of unit records; thus, information on manpower in hospitals could be combined with information on patient hospital stays. Linkage of such data bases will enable the health planner to quantitatively describe the health care system in order

* CHSS are Statewide systems funded in part by the National Center for Health Statistics/Department of Health, Education, and Welfare.

TABLE II

Data Item	SSA Form 2874 (UHDA)	PSRO Data (PHDDS)	Div. of Health (HSDS)	Form 1453 (Bill)	PAS	Induced Abortion	Hospital
Provider Name				XX	XX	XX	XX
Address (City, County & State)				XX		XX	XX
Number (Assigned by Medicare)	XX	XX	XX	XX	XX		XX
Patient's Name	XX		XX	XX			XX
Address (City, County & State)	XX	XX	XX	XX	OPT	XX	XX
Zip	XX	XX	XX	XX		XX	XX
Social Security Number	XX	XX					XX
Claim Certificate I.D. No. Medicare	XX			XX			XX
Claim Certificate I.D. No. Medicaid	XX			XX			XX
Medical Record Number	XX	XX	XX	XX	XX	XX	XX
Date of Birth	XX	XX	XX	XX	XX	XX	XX
Sex	XX	XX	XX	XX	XX		XX
Race	XX	XX	XX		XX	XX	XX
Marital Status			OPT			XX	XX
Educational Level						XX	XX
Admission Date (Mo., Day, Yr., Hr.)	XX	XX -	XX	XX	XX		XX
Nature of Admission		XX	XX		XX		XX
Admitting Diagnoses				XX			XX
Admitting Service			XX		XX		XX
Site Admitted From			OPT		XX		XX
Expected Principal Source of Payment	XX	XX	XX		XX		XX
Secondary Payor	XX			XX			XX
Dates of Qualifying Stay				XX			XX
Certification Status		XX			XX		XX
Days Certified Initially		XX					XX
Attending Physician	XX		XX	XX	XX	XX	XX
Social Security Number	XX	XX					XX
Operating Physician	XX		XX		XX		XX
Social Security Number	XX	XX					XX
Consultations (Number)			OPT		XX		XX
Type or Identity			OPT		XX		XX
Patient History					XX	XX	XX
Diagnoses (Principal)	XX	XX	XX	XX	XX		XX
Other	XX	XX	XX	XX	XX		XX
Coded Diagnoses	XX		XX	XX	XX		XX
Procedures and Dates	XX	XX	XX	XX	XX	XX	XX
Coded Procedures	XX		XX		XX		XX
Anesthia			OPT	XX	XX		XX
Tissue Report			OPT		XX		XX
Complications			OPT		XX	XX	XX
Discharge Date (Mo., Day, Yr.)	XX	XX	XX	XX	XX		XX
Discharge Service			XX				XX
Number of Requests for Extension		XX			XX		XX
Total Days Certified		XX			XX		XX
Disposition of Patient	XX	XX	XX		XX	XX	XX
Transfusion			OPT	XX	XX		XX
Accomodation (Bed)			OPT	XX			XX
Special Units Used			OPT	XX	XX		XX
Therapy -- Drugs				XX	XX		XX
Lab & Clinical Tests & Exams (temp., blood pres., hematology, function, etc.)				XX	XX		XX
Other Services (pharmacy, radiology, physical therapy, etc.)				XX	XX		XX
Charges for Services			OPT	XX	OPT		XX
Deductibles				XX			XX
Date Guarantee of Payment Began				XX			XX
Date UR Notice Received				XX			XX
Date Active Care Ended				XX			XX
Date Benefits Exhausted				XX			XX
Lifetime Reserve Days Used				XX			XX
Non-Covered Days				XX			XX
Covered Days				XX			XX
Date of Preparation	XX						XX
Signature of Provider's Representative	XX			XX			XX
Patient Authorization to Release Info				XX			XX

to plan for future allocation and development of resources.

It will also be possible to conduct many different types of studies and employ a variety of analytical techniques for monitoring of the nation's health services. For example, in Professional Standards Review Organization (PSRO) program data is collected from the hospital medical record. This involves abstracting from the record of each patient information such as diagnosis, age, length of stay, etc. Once a national program is under way and data is available on all discharges, it would be of interest to examine possible correlations involving environmental or demographic variables with PSRO data such as length of stay, frequency of diagnosis, morbidity and mortality. Since geographic and demographic information is not available from the PSRO data base, the ability to link with data bases generated by other programs such as the Health Resources Administration's (HRA's) Hospital Discharge Survey (HDS) and the Area Resource File (ARF) is necessary for conducting meaningful epidemiological studies.

DATA BASE DESIGN

In attempting to advance the health care services of our citizens, data collected has many roles to play. It can be used for monitoring, evaluation, information exchange, research and determination of policy. As such, we can expect a great deal of separate data collection and processing by the various government agencies. However, we expect a significant portion of the data elements in the separate data collections to overlap. The advantages of having a common coding system for the data elements and of structuring the elements into data bases which can be linked are self-evident. To achieve such a goal requires multiple planning in that the original work of the respective agencies must be collated, reviewed and restructured. The same holds true for the establishment of procedures and standards for data collection.

In structuring such a system, one must consider necessary features of the data base. The volume of data will be very great. In order to access and manipulate the data rapidly, the system must be modular. Separate files must be self-contained with respect to special topics. Thus, there can be a separate manpower file with geographical regions forming its subfiles. However, the file must contain crosslink indicators to all other relevant files and alternative classifications. If all of this is done, the data will be widely probed and unforeseen uses and requests will arise.

The data base design must be justified as an element of an information system. The criteria for rational design of information systems must be based upon the needs of the users in the performance of their activities. Information systems or services which are not designed on this basis will inevitably fail to meet user's needs. A logical first step in the analysis of the adequacy of existing systems and in the design and implementation of new ones, is the determination of the functions and activities of the potential users. This first step allows the determination of other needs.

The definition and enumeration of categories of use for better care includes:

- Policy, planning and program management and resolution at the Federal, State and local levels.
- Research at all national institutions.
- Application, i.e., the community which translates research results into practical applications.
- Implementation, i.e., those groups of individuals or organizations who are directly responsible for initiating health care actions.
- Dissemination, i.e., ensuring that the information is made known to those who may need it.

As an example, a health warning system may be set up as a major use of the data collected. The large amount of data collected will allow for the establishment of meaningful norms and expected deviations. It will be possible to use the data to establish baselines and look out for trends. Based on the deviations from established norms the impact of a given disease on an area could be determined.

In the massive collection of data the data elements must be standardized and organized into identically formatted records at the various source collection points. If this is not done, large processing costs will be incurred in attempting to use the data. However, these eminently practical procedures put a damper on the use of the data collected for creative investigation of medical questions. Let us consider an example. There is reason to believe, based on international evidence, that cancer of the colon is caused by either large meat consumption or low consumption of cereals.[1] This is established by showing the correlation between the incidence of cancer and meat consumption in 23 countries. The data are adjusted to eliminate differences in age distribution in the population. The possibility exists that it is not meat consumption, per se, but some other factor associated with meat consuming countries which gives rise to the correlation. A question worth answering in this regard is with respect to the United States population: "Does higher meat consumption appear to be a causitive factor in cancer?" To answer this question, it is necessary to obtain data from people whose environment and life styles are very similar except for their meat-eating habits. We can achieve our goals by carrying out physiological studies and comparing the hospitalization records of vegetarians with a socio-economically equivalent group of meat eaters. Here is where the catch is. This is not the sort of information that would be entered as part of the standard records collected on health care (and it should not be routinely collected). This means that a special effort must be made to obtain this information, most likely surveying the former patients for this one detail. However, the standard records that comprise the data base will, in our suggested record design, allow for a preliminary search to determine the target population for the survey. They will also contain most of the auxiliary information that is required; thus limiting the data collection needs. A very striking example is the case of the clinical trials which were planned to see whether people can voluntarily decrease their risk of heart disease by watching their diet carefully; in particular, whether reduction of cholesterol will have a significant effect on decreasing the risk of heart attack.[2] In attempting to carry out the clinical tests large sample sizes were needed in order to obtain a statistically valid analysis. The initial costs were expected to be 80 million dollars. Due to inflation and unanticipated costs, these trials will cost at least 200 million dollars. A large part of the cost is due to the large population sample that must be taken in order to assure that enough subjects within the sample will develop heart problems during the study period. If we consider the case where we have a large data base, we can choose the target population from out of our data base records and conduct a retrospective study. In this way, the information may be obtained at a much lower cost. Since retrospective studies are never as good as studies which are monitored from the start, another possibility is that the retrospective studies could be used to supplement clinical studies of a smaller scope; the combination of both types of studies will lend to greater validity of results. In addition, the retrospective study can be used to characterize the patient type who is more susceptible to a given type of disease and, in this way, allow for better clinical screening reducing the need for large samples in certain clinical studies. All of these possibilities exist. To what extent they can be implemented with the use of mass data collection depends on the proper planning and layout of the data elements to allow for such usages.

Within recent years the Federal Government has been heavily involved in efforts to standardize data sets based on hospital stay. In 1971, the Department of Health, Education, and Welfare (DHEW) examined the possibility of requiring a uniform set of data on all federally funded discharges from short stay, acute care hospitals. The Uniform Hospital Discharge Data Set (UHDDS)[3,4] was designed as the minimum data set, uniformly defined, capable of providing all users basic and comparable information on all hospital discharges. In 1972, the Uniform Hospital Abstract Form Subcommittee of the U.S. Committee on Vital and Health Statistics recommended the minimum basic data set. Minor modifications of this data set were made in 1974 for use in DHEW programs. The modified UHDDS with definitions is presented in Table III.

TABLE III—Uniform Hospital Discharge Data Set (UHDDS)

Person Identification
Each admission is to be reported by the patient's unique Social Security number.
Date of Birth
Sex
Race
Residence
Hospital Identification
The provider number assigned by the Medicare Program and used by Medicare and Medicaid in the hospital certification process.
Admission Date and Hour
Month, day, year and hour of admission
Discharge Date
Month, day and year of discharge
Attending Physician
This is the physician primarily responsible for the care of the patient from the beginning of this hospital episode.
Operating Physician
This is the physician who performed the principal procedure.
Diagnoses (Principal and Other)
Procedures (Principal and Other)
Disposition of Patient
a. Transferred to another short-term general hospital
b. Discharged or transferred to skilled nursing facility (SNF)
c. Discharged or transferred to an intermediate care facility (ICF)
d. Discharged or transferred to another institution
e. Discharged to home or self-care (routine discharge)
f. Discharged to home under care of an organized home health service.
g. Left against medical advice.
h. Died
Expected Principal Source of Payment
a. Self-pay
b. Workmen's compensation
c. Medicare
d. Medicaid
e. Other Government Payment (e.g., CHAMPUS)
(1) Title V
(2) Other
f. Blue Cross
g. Insurance Company
h. No charge (free, charity, special research, or teaching)

The long range goal is to have all users of the UHDDS, including PSROs (explained in Reference 5), claim payment agencies, State Cooperative Health Statistics Systems, discharge abstract services and planning agencies, receive these data through the use of a standard form and standard tape format. This approach would substantially reduce the data collection efforts within hospitals and provide all users with consistent, compatible data to be combined with data from other sources to meet total data needs.

At the Federal level several agencies have similar data requirements; however, at present there is no method of coordinating and linking these data bases for use by any agency. There is no reason, in principle, that one agency could not collect and disseminate all the required data. However, many different sub-agencies have their own mission and legislatively ordained functions to follow. As a result the entire array of independent requests for data from different agencies arises. The data collected under present conditions will form independent data bases which have a large amount of redundancy, but cannot be used in conjunction with other data bases in any simple manner. Incompatibility of the data files and data bases collected by different agencies will make it very costly to retrieve information with cross-agency applicability. The piecemeal specification of data bases is almost guaranteed to cause enormous duplication effort with resultant products of limited use. At the very least a general systems plan should be formulated to lower costs of collection, storage, software development for analysis and processing cost. Consider, for example, the computerized health resources information system known as the Area Resource File (ARF) which has been developed for the Bureau of Health Resources of DHEW. The County file unit identifier uses the Federal Information Processing Standards (FIPS) code for State and county locations. This sounds like an excellent means for standardization. However, in order to provide for the possibility of analysis of the data according to different ways of grouping counties, the following additional area identifiers are included:

1. State Economic Areas (SEAs) and Economic Subregions
2. Standard Metropolitan Statistical Areas (SMAs)
3. OBE Economic Areas
4. Federal Regions
5. Ranally Major and Minor Areas
6. DMI Place Sizes
7. Comprehensive Health Planning Areas
8. Professional Standards Review Organization Areas

With all of these identifiers only classifications which group whole county units are meaningful. In addition, the ARF is under revision with a contract out to restructure the basic file and add new elements. Among these new elements are further area identifiers such as the DHEW region code. It is easy to see how with time identifiers can keep on proliferating.

SYSTEM DESIGN CONSIDERATIONS

Since we must deal with many uncoordinated ongoing activities, the first requirement is to discover what present data collection and storing is related to health and what is planned for the future. Each of these activities must be evaluated in terms of its own mission and the potential meaningfulness of its interaction with other activities. Some of the factors which must be taken into consideration are:

1. *Collection of Data*
   Quality
   Reliability
   Duplication
   Meaningfulness of relationship to other data collection
   Cost
   Value
   Burden of collection on institutions.
2. *Processing of Data*
   Error checking
   Avoiding redundancy
   Assuring timeliness
   Ensuring availability and transferability
   Analysis of data
   Establishing norms and checking deviations
   Determining policy and operational impact
   Investigating linkage
   Determining level of usage of data

An approach to coordinating and making the collected information accessible through the system structure is by means of a hierarchical catalog. The catalog will contain information on all large data bases which involve interface uniformity. At the peak of the hierarchy are all the most generalized controlling data summaries, which may have a series of underlying data bases to support them. Thus, we might consider a possible chain of data bases each at a more detailed level:

- DHEW
- BQA
- Local PSRO
- Hospital
- Doctor (Patient)

Top level summarized catalog information may also refer to a segment of an independent data base with a different orientation. Thus the budget data of the Office of Management and Budget (OMB) summarizes all Federal funding of medical services. Another example would be analogous to the present computerized Program Review and Evaluation System of the Environmental Protection Agency (EPA). This program compiles in one location information on all monitoring programs operating throughout EPA. The control file can then be used as a starting point to identify and locate any specific monitoring operation. In our case the file would record all environmentally oriented data bases which have a health care impact.

The data base catalog at the top level would contain in its enumeration of data bases such information as the organization or agency and/or person in charge, the descriptors of the file structure, the sub-files (if any) that the data base derives from, the period covered, the area involved, and the major intended use.

At each succeeding level there will be a directory of the major segments into which that level is divided. Each "layer" performs its own set of operations and interfaces with levels above and below. Basic keys in each level allow for interfacing. The keys and links allow full reformulations of the data bases for special purposes, such as merging with another file. By keeping the data structure simple it allows for each creation of a new special purpose data bases with minimal programming. Note, the hierarchical classification and its tree structure refer only to the organization of the data bases themselves and not to the contents of the data bases. Insofar as possible, relational information should reside in the data itself or be explicit. It should not reside in the structure itself, for then two different data bases could not easily be combined.

A review of the present data gathering activities and existing data bases should lead to the definition of an overall system structure. All elements of this conceptual system must then be evaluated in terms of their aims and interactions with other parts of the system. The critique should lead to some degree of redesign with emphasis on factors such as the following:

- All data bases shall possess the flexibility to adapt to new requirements.
- The data base system shall be modular in nature in order to provide for future processing capabilities.
- The software support systems shall be:

(1) high-level language
(2) relatively machine independent

The data should have linkages to allow for the merging of subsets of different data bases. Planning must include flexibility which in the case of the data base means reserving unused pointers for future linked list expansions.

HUMAN INTERFACE WITH SYSTEM

A data base which covers a large segment of the population can give statistically valid answers to questions addressed to the data. For example, in the case of the elderly their insurance by Medicare ensures that most illnesses causing hospitalization will be recorded and reported. To the extent that their illness has been properly diagnosed, encoded and stored in the data base, one can obtain interesting profiles of the relative frequencies of major illnesses of the elderly. This information in turn can be distributed to local health planning agencies and used within the Federal Government to better determine resource allocation. However, these uses, as shown in our general analysis, depend on having responsible people scanning the data and taking action where necessary or passing the information to those responsible to act. There is no way of guaranteeing that proper action (or any action) is taken; however, in the design and description of human interactions which will insure that the large masses of data are reviewed and channeled to the proper agencies for appropriate action.

In terms of the data we wish to achieve compatibility and transferability of the data for use in the different agencies. The larger question we face is how to ensure that appropriate data will be used and shared and acted upon. This depends upon both the system and the people. Responsible and authoritative personnel in key positions can act as control points to spot unusual occurrences and optimize ordinary workflow. However, the fragmentation of the governmental health establishment (DHEW) makes meaningful flow of data difficult.[6] A coordinating and controlling body is therefore vital to support and expedite data transfer activities. In fact, DHEW requires a coordinating mechanism which will allow for review of all the data collecting needs and requirements of its various agencies.

Data input to the information system must reflect the realities affecting the providers of the data, i.e., physicians admissions officers, record librarians, fiscal intermediaries and patients. Requirements or requests for more data may not be relistic unless they bear directly on improving patient care and providing better health services.[6]

Information systems are often designed in the vacuum of the bureaucracy in a manner dictated by logic and efficiency. This approach is acceptable in the technological environment of the business world, but in the humanistic world of medicine, the patient-physician relationship determines and moderates the exchange and availability of various types of health data.

Concomitant with the input of data to the system is user access. Individuals must be able to determine where the data of interest resides within the system, who to go for access and how to use the information obtained. The overriding concern relevant to both input and access is confidentiality of the data—the protection of the physician-patient relationship from the potential abuse inherent in a massive system capable of linking various data sets into single comprehensive records.

As stated, we are dealing with sensitive medical data where human judgment is essential at all steps in the routing and processing of the data. Therefore, the controlling body will have to be supported by a large network of people sensitive to the needs of the medical community. At each point along the way these responsible persons will try to flag vital aspects of the data and bring it to the attention of those who might act upon it. It is essential that the individuals responsible for evaluating data have relevant backgrounds to make such evaluations. For example, as information comes into an organization and is analyzed by some statistical group whose function is to establish, maintain or review medical norms for various sections of the country, one of the people who looks over the work of the group must be familiar enough with norms and medicine so that if an unusual deviation occurs, he will alert the local medical authorities of such a development and appropriate action will be taken. This might be considered to be analogous to the Center for Disease Control's monitoring of potentially epidemic diseases on a national and regional basis. If someone at a higher level notices a long term range increase in the incidence of a given disease, unless he is in the position to initiate action, the data will merely be published in some official bulletin and set to rest in a convenient archive. It is important that throughout the system there are checkpoint people who make sure that when data is important, it is routed, flagged and given high priority; when it is not important they will suggest the removal of data from the system that need not be collected. What this amounts to is that with the planning of the hardware and collection network there also must be planned a software network of people, so that not only is the normal flow of data expedited in the ordinary manner, but action outside of the normal flow can be taken.

SUMMARY AND CONCLUSION

Once established, the system could provide health planners, epidemiologists and other researchers with the means to undertake studies of a type and magnitude not heretofore possible. For example, PSRO activities will result in the establishment of a national data base containing unit record discharge data on all Federal discharges from short term acute care hospitals across the nation. The ability to link this health data geographically with data contained in the Environmental Protection Agency and Department of Labor or other Federal data bases on industrial pollutant levels would allow epidemiological studies of the relationship of pollutants to hospitalization, disease etiology, frequency of surgical procedures, etc.

Linkage of hospital utilization data with the geographically coded Area Resource File data bases containing facility and manpower data could provide information over time on trends in manpower and facility availability relative to hospital utilization; providing health planners with the means to predict the need of any given area in the country and make appropriate recommendations for future facility construction and health manpower allocation.

Once established with the Federal health system, the utilization of an information system capable of linking data from many sources is limitless. However, the system is only as good as the people running it. It is essential that all along the flow of data through the system that human checkpoints be established to monitor and appropriately act upon the output of the system.

REFERENCES

1. Cairns, J., "The Cancer Problem," *Scientific American*, Vol. 233, No. 5, 1975.
2. Murnaghan, Jane H., "Health Service Information Systems in the United States Today," *The New England Journal of Medicine*, Vol. 290, No. 211, 1974.
3. Murnaghan, J. H. and K. L. White, (Editors), *Hospital Discharge Data, Report of the Conference on Hospital Discharge Abstracts Systems*, Lippincott, 1970.
4. Hodgson, D. A., L. E. Kucken and J. M. Ensign, *The Uniform Hospital Discharge Data Demonstration Summary Report*, DHEW (HRA-74-3102), 1973.
5. Goran, M. J., J. S. Roberts, M. A. Kellogg, J. Fielding and W. Jessee, "The PSRO Hospital Review System," *Medical Care*, Vol. 13, No. 4, 1975.
6. White, K. L., "Priorities for Health Services Information," *Health Service Reports*, Vol. 88, No. 22, 1973.

Data base processor technology

by DONALD R. ANDERSON
Sperry Univac Defense Systems Division
St. Paul, Minnesota

ABSTRACT

The ability to achieve significant amounts of mass memory has not been matched with a capability for processing data stored within the mass memory. The concept of a dedicated data base processor that would implement microprogrammed data base primitives with high speed microprocessor technology appears to be a plausible means of obtaining higher performance data base processing systems. The premise of this concept is that simple operations that manipulate data base information can be included in a separate, simply structured processor that operates in parallel with the central system. In doing so, parallel execution benefits are realized that could lead to higher system performance levels.

The technology that supports the dedicated data base processor concept, system architectural considerations, the application of the concept to both distributed and centralized systems, and areas of needed research are described in this paper.

INTRODUCTION

Memory technology developments are providing an increasing number of alternatives for achieving significantly large amounts of system storage at differing levels of performance and cost. These alternatives presume conventional system architectures which are not optimized to permit higher performance levels based upon the parallel processing that is felt to be inherent to data base processing problems. Newer memory devices, Charge Coupled Devices (CCD) and Bubble memory, are taking their places along with improved disk capabilities and the newer cartridge units to provide for storage of 10^9 bits or better but these memory technologies seem deficient in that while they extend the system mass memory capacity, they do not provide additional system processing ability.

This paper describes the concept of a Data Base Processor (DBP) which specializes data base processing functions within a processor that is closely associated with mass memory. These processing functions are normally time-shared on the central processors with other user functions. The data base processing concept recognizes that a system containing significant mass memory has a dominant part of its resources expended toward processing mass memory data and that specializing data base processing functions in a separate and independent processor could yield substantially improved system performance.

THE DBP CONCEPT

There are many data processing applications where the data processing system is designed to process information stored on mass storage devices in the form of a regularly structured data base. These applications differ from command/control or process control in that significant amounts of data move to/from mass storage devices in the normal processes that occur during system operations. Data base systems also have less stringent real-time (i.e., second versus millisecond response times) requirements than the process control or command/control systems.

In applications where data base operations predominate it appears reasonable and profitable to consider an alternative to conventional system architecture to provide more balance between lower-level mass memory access/processing operations and data base information manipulations to achieve higher system performance. The DBP architectural concept is to move a subset of the overall data base processing away from central processing facilities into a processor that is intimately associated with the mass memory upon which the data base is stored (Figure 1) and which has only high-level function characteristics (instructions) for data base processing. In effect, a parallel processing architecture is created in which memory access/processing operations are performed in parallel with central processing operations.

The advantage of the DBP architectural concept can be illustrated by considering the processing of rèlational data base information. In a conventional architecture, relation content processing is performed by the central processor and intuitively, there are instances where the entire relation content is not needed, especially where decisionary processes that select or reject data are involved (as in the case of linked files). The access to, movement of, and subsequent examination of the relation content represent unnecessary sys-

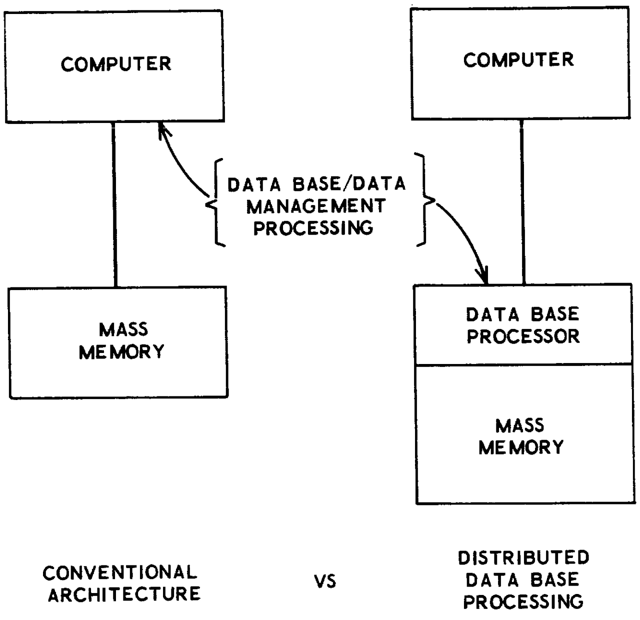

COMPUTER
MASS MEMORY
DATA BASE/DATA MANAGEMENT PROCESSING
COMPUTER
DATA BASE PROCESSOR
MASS MEMORY
CONVENTIONAL ARCHITECTURE
VS
DISTRIBUTED DATA BASE PROCESSING

Figure 1—Distributed data base processing concept

tem overhead if the end-object desired is not in the relation accessed. A data base query, for example, that builds a list of all employees who are female would require, in a conventional architecture, the movement of an entire relation content into the central processor even though only part of the relation is needed in order to establish the desired list. Contrastingly, in the postulated DBP architecture concept where the processing of the relation content occurs away from the central processor, the only information that is provided to the central processor is that which is the result of processing the relation data. This reduces central processor loading allowing more central processor time for further processing of the file information.

Linked files are another example where the DBP architecture concept would provide higher performance. With linked files the desired information is accessed through a series of files each of which contains information that, in effect, points to or links to succeeding information. Ultimately the decision process links to the end-object desired. In a conventional architecture each succeeding file must be moved into the central processor for examination, which requires physical movement of the file into the central processor, synchronization with concurrently executing tasks, binding the data to the process which examines the file data, and then initiating access to the succeeding file should one be desired. With the DBP architecture concept, most of this could be eliminated by performing the search and decision process within the DBP.

Conceptually, the DBP is a semi-independent, simply structured, special-purpose processing unit that operates under high-level control of the central processor. The DBP can be interconnected with a central processor in any of several ways discussed later and is interfaced with mass memory as dictated by the mass memory interface characteristics. The DBP contains data base processing primitives to which it is directed by the central system. When executing its own program, the DBP operates independently of the central system, accessing and using mass memory facilities as directed by its internal program. The DBP contains its own independent buffering facilities for holding data base information to be processed and is programmed as a special-purpose microprogrammed processor with separate microprograms implementing high level processing functions.

DBP TECHNOLOGY FACTORS

There are four technology factors which support specializing data base processing functions: Distributed Processing, Data Base Languages, Microprocessors, and Mass Memory Technology. Each of these factors contributes to the overall concept, but none completely provides a total solution. When supplemented with additional research, however, these factors indicate feasibility of the DBP concept.

Distributed processing

Functionally distributed processing principles seem applicable to the DBP concept. Microprogrammed data base processing functions are separate and independent processes executing in specialized processing facilities[1] (i.e., functional distribution).

Data base languages

High level data base processing languages form a uniform and consistent method of expression through which data base processing problems are solved. The syntax and semantics of these languages incorporate subprocesses which, when partitioned, resemble primitive functions that could exist as microprogammed processes in a data base processor.

Microprocessors

Economical integrated processing logic is available at very attractive performance levels.[2] Many special purpose functions can be implemented with significantly improved execution times.[3]

Mass memory technology

Through magnetics, CCD,[4] and Bubble[5] memory, mass memories of greater than 10^{11} bits may be built. An intelligent memory which would combine on-chip processing with semiconductor logic would extend to the data base processor concept.

SYSTEM ARCHITECTURE CONSIDERATIONS

Distributed and centralized systems

The data base processing concept applies to both distributed and centralized processing systems. The implementation may differ with the architecture into which it is merged, however, the basic principles of the idea remain unchanged. A distributed system is a system of independent computers interconnected with a mechanism for information transfer between the independent computers which may be a packet switched communications network or a specialized ring structure.[6,7] A centralized system is a conventional unit computer or multiprocessor system whose cental processor facilities are not physically dispersed. Remote terminals may be located away from the central processor in a centralized system.

The architecture principles of the data base processing concept that apply to both distributed and centralized systems are the at-location processing of data base information which lessens the central processor overhead and the implementation of a system-parallel data base processing capability. In a network distributed system, communications loading would also be lessened.

In distributed systems where the communications network utilizes common carrier lines (50 kb/sec line interconnections, for example) the DBP could interface with a central processor which would provide it with the high level commands (Figure 2). The central processor in this instance provides the necessary interfaces with the network protocol, the translation of requests from users into DBP commands, the translation of DBP responses into network format, and the return of the network formatted data to the user. In instances where the interconnecting network is a special or dedicated structure (Figure 3) or where it is desirable that the DBP directly interface with the network, the DBP would interpret protocols and multi-user data base inquiries in addition to the retrieval and manipulation of the data base information.

Memory hierarchies

The practical achievement of a significant mass memory is governed by economics. The most plausible and practical way of implementing a mass memory today is through a combination of technologies which implies a hierarchy of mass memory storage and the ability to control the movement of information between hierarchy levels.[8] The DBP, if it is assumed to be used with a hierarchical mass memory system, must provide for information movement between various levels of the hierarchy and for handling interfaces unique to the different mass memory devices used.

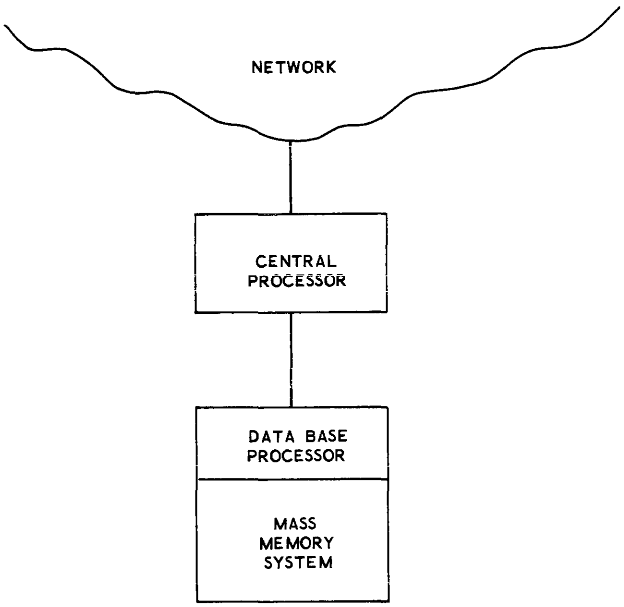


Figure 2—DBP network interconnection through host

DBP-hardware interfaces

The general architecture design of the DBP must accommodate three classes of hardware interfaces—network, central processor, and mass memory—in order to be applicable to a wide variety of system architectures. In the network system, when the DBP assumes

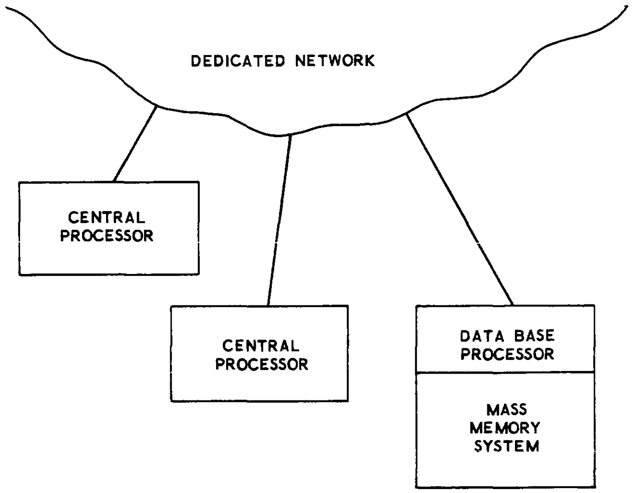


Figure 3—Direct DBP network interconnection

a host-equivalent role it must also assume the interface responsibilities that a host assumes. This includes physical hardware interfaces (serial, parallel, rates, coding format, control signals) and the ability to act as a receptor/transmitter and interpreter of message control (protocol) information. The network interface problem appears to dictate a multi-form network interface module that can be used for interfacing with either special purpose interconnects or general purpose networks. Different versions of this module would provide a standard internal DBP interface.

Two classes of central processor interfaces are needed in the DBP architecture: I/O channel and DMA. The I/O channel interface seems to be the most important considering the general philosophy of the DBP architectural concept. However, a DMA capability coupled with suitable memory protection could provide a means of common memory communication between the DBP processes and the general data management processes which are performed by the central processor. The I/O channel interface should be capable of handling multiple users, either as a multiplexed single channel, or as separate physical I/O channels multiplexed internally within the DBP. The DMA interface with the central processor should incorporate memory protection controlled by the central processor. This is especially needed when virtual environments are provided by the central processor.

Future mass memory technologies that can be expected to be used with the DBP are block organized; discs, MNOS BORAM, CCD and Bubble memory devices, magnetic tapes, magnetic cartridges, etc. The DBP architecture must be able to accommodate, through interface modules, different mass memory interfaces. In the case of some mass memory technology, disc technology in particular, the interface module could be identical to the central processor interface module. Support of the mass memory interface through microprogramming appears to be a requirement.

DBP ARCHITECTURE REQUIREMENTS

The DBP system requirements form a general bound for the DBP design. In a sense, the DBP extends data base management to a parallel processing environment in which one of the processors is specialized for independently processing data base information.

Modular structure

The DBP should be organized in modular fashion (Figure 4) in order to accommodate different interfaces and be usable in different system architectures.

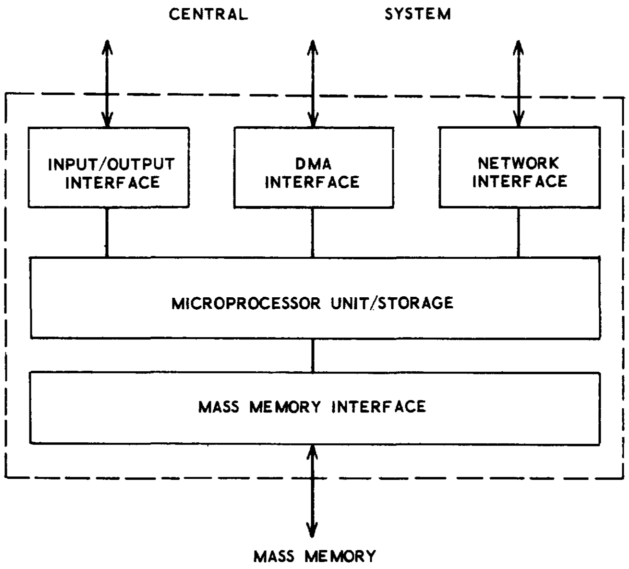


Figure 4—DBP modular structure

Primitives

The DBP shall implement microprogrammed primitive functions that can be used by the operating system or data management system for processing of file data concurrent with central system operations.

DBP-internal processing

The DBP shall be capable of independently processing data stored on mass memory or in its own internal memory.

Multi-operation processing

The DBP shall be capable of performing a limited number of operations concurrently to permit efficient usage of the DBP where interfaced with mass memory with significant latency times.

Memory hierarchy

The DBP shall contain the necessary interfaces and control programs to support a mass memory hierarchy and to transfer information between hierarchy levels.

Rate/format matching

The DBP shall provide the buffering/processing required to match mass memory intrinsic speeds and formats to central system transfer capacities and formats.

MAJOR DBP ELEMENTS

The DBP system architectural concept uses specialized, independent microprocessing logic to perform data base processing operations. The elements needed to implement the DBP modular structure are described in preliminary fashion to indicate hardware technology that could be used.

Processing element

The DBP processing element will be a 16-bit, high speed (200 ns cycle time), microprogrammed unit with separate micromemory and buffer storage. Internally the processing element will be an asynchronous bussed structure.

Buffer storage

The DBP will have a word organized, high speed buffer memory (4,096 words) for command storage, intermediate storage of file data, and rate matching between mass memory and central system interfaces.

Input/output interfaces

The DBP will have both byte and word parallel I/O interface modules capable of multiplexed and nonmultiplexed operations.

Mass memory interfaces

The DBP will support wide word (in excess of the DBP internal word width), narrow word (less than DBP internal word width), and serial mass memory interfaces.

DISTRIBUTED DATA BASE SOFTWARE DESIGN

Implementation of the DBP concept requires rethinking of the major system software elements. The DBP concept is intended to relieve conventional system processors of their workload and in doing so, modifications of existing software structure must be anticipated. At this time, three software-related problem areas have been identified. These are the modified operating system software structure, the definition and design of microprogrammed primitives, and user control of the primitive operations.

Operating system software/DBP interfaces

A simplified diagram of a conventionl data base management software structure is shown in Figure 5. Conventionally, the operating system interfaces directly with secondary storage on which the data base is stored to move data between the secondary storage and the system buffer allocated in the computer main memory. User programs and the Data Base Management System do not interface with secondary storage but with the system buffers.

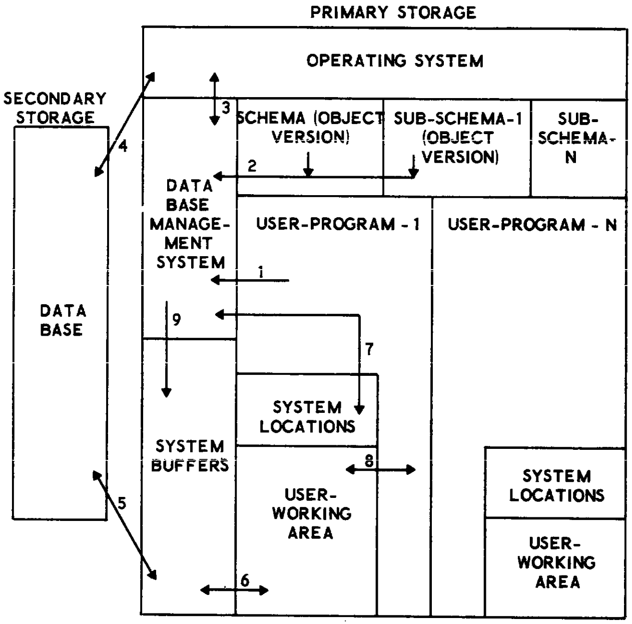


Figure 5—Conventional DBM software structure

Incorporation of processing capabilities within the secondary storage poses questions of which software elements should directly relate with and access the data base. The existence of processing primitives at the data base in effect causes the user program to transcend the primary store-secondary store interface; i.e., the execution of a user program takes place in two places with possibly intervening software elements that do not necessarily permit direct communication between concurrently executing parts. The enumerated paths provide a depiction of the operations which transpire in the conventional software structure:

"1" a call for data by the user program
"2" DBMS supplements the call arguments with schema information
"3" DBMS requests I/O operations from the Operating System (OS)
"4" The OS interacts with secondary storage
"5" The OS transfers data between secondary storage and system buffers
"6" DBMS transfers data between system buffers and the program working area
"7" DBMS provides status information to the calling program
"8" Data in program's working area manipulated as required using host language
"9" DBMS administers the system buffers

Primitives implemented in a data base processor would modify the DBM software structure (Figure 6). In this proposed structure, operations would:

"1" a call for process execution by the user pro-

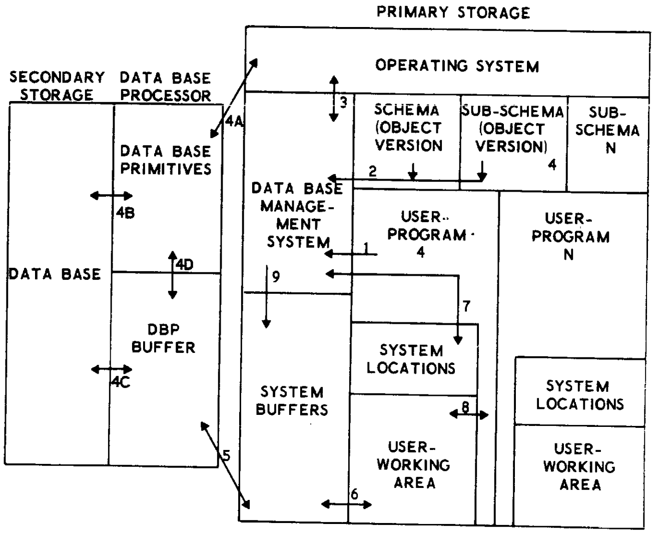


Figure 6—Modified DBM software structure

gram. This extends the original structure concept by including the capability of the user program to request a process to be executed and relying on the DBMS to formulate and control the requested process. Previously this was limited to a call for *data* which eventually was manipulated by the user program.

“2” DBMS supplements the call arguments with schema information.

“3” DBMS requests I/O operations from the Operating System (OS). In this operation, modifications may have to be made to expand the I/O function to include a differentiation between DBP function commands to be sent to the DBP (or status information requested from the DBP) and normal buffered data transfers.

“4A” The OS interacts with the DBP. The DBP acts as the OS/DBMS agent to perform, through execution of primitives, the operation requested.

“4B” The DBP interacts with the secondary storage to perform, in association with its internal storage, the operation requested.

“4C” The data base information is transferred from secondary storage to the DBP buffer for processing and returned to secondary storage should it be required.

“4D” The DBP accesses its internal storage in operating on the data base information.

“5” The OS transfers data between the DBP Buffer and the System Buffer. To the OS no change should be apparent during this step. The secondary store/DBP would appear as a buffered secondary storage system.

“6”–“9” No major changes apparent in these operations.

The DBMS, rather than the OS, software design would be impacted the most by inclusion of DBP primitives. These impacts would center heavily on including capabilities to operate a concurrently executing process in parallel with user program execution. The overall DBMS control strategy and methods definitely require further research.

Definition of microprogrammed primitives

The intent of the DBP is to provide improved system performance through the use of microprogrammed primitives. The definition of the primitive operations to be performed is difficult. User program needs, DBMS needs, and the practical capabilities that can be incorporated into hardware all impact the primitive function definition. Considerations in defining the primitive function set include:

General purpose vs special purpose

A truly general purpose set of primitives that benefit all possible user programs is probably unattainable given realistic development resources. A totally special purpose primitive function capability, on the other hand, would create an unnecessary burden on many users because they would be required to develop, or modify for use, the basic operations needed to interface with secondary storage. The eventual solution would appear to be a compromise between these extremes implementing a selected set of general purpose primitives that would be useful in many user applications and supplementing that general purpose set with a user capability for creating special purpose functions that are of immediate and direct benefit.

Dynamic vs static primitive function sets

Dynamically changing machine instructions (primitives) that make multiple usage of the same logical operation codes create system state management difficulties. Dynamic primitive function sets are desirable in processing environments that presumably service many different users. Environments which tend to be application-static do not appear to require dynamic function sets.

Direct HOL vs machine level operations

Direct implementation of DML (or other data base languages) elements through microprogramming represents an approach to the design of the primitive function set. At the other extreme, the primitive function set could be defined to encompass the functionality of conventional machine instructions. The necessary properties of the primitive function set seem to be to provide the user the freedoms of high level code, but also allow him, at some time during the system development and design process, machine level specification of the primitive processes.

Data base management vs file processing

Data Base Management is the organization and management of the system of information rather than the manipulating and processing of data base information subsets (File processing). User programs are most probably concerned with the latter, whereas the Data Base Management System is concerned with the former. Neither should be neglected in the definition of the primitive function set. The DBMS, in managing large data bases with a significant number of uniquely identified information elements, encounters difficulties analogous to the difficulties encountered by user programs, namely movement of files between mass storage and primary storage, processing of those files, etc.

Mixed language primitives

A choice exists between defining primitives which are total process entities (i.e., single user statement, single process) and defining user-apparent primitives that are a mixture of interpreted language statements and microprogrammed processes. The mixed-language primitive approach supports standardized primitives but still retains primitive customization capabilities.

Standardized relations

Standard file manipulation relations have been defined for file processing. These include cartesian product, union, intersection, projection, diadic restriction, monodic restriction, join, composition, permutation, computation, difference, inversion, and ordering and are representative of standardized functions that would form the basis of a primitive function set. Other standardized relations for moving data between hierarchies should also be considered.

User control of primitive operations

There are two kinds of users of the DBP. One is the application user represented by the user programs in the machine. The other is the systems programmer who is represented by the DBMS/OS software. DBP primitives should support both users. How these users control data base processing operations through primitives depends to a large extent on the kind of computer system operating environment they are operating in. In very specialized systems where the applications code and the system software are part of an integrated design, it is possible to have application user programs directly command and sequence primitive operations. In general purpose systems where predesigned operating systems and data base management software exist, control responsibilities would probably be relinquished to the DBMS. The specific choice of how the user controls primitives remains a function of the operating environment.

A common problem that will be encountered by both kinds of users will be the control of parallel execution. The DBP, as an independent processor, represents a specialized computation resource which can operate in parallel with other processors. Computation results from the DBP must be merged with program main stream which requires that methods of controlling parallel processes be established. To some extent these methods have been researched for multiprocessors. At the applications user level, however, DML languages have as yet no provision for invoking, staticizing, or merging parallel processes. At the system user level, the DBMS will require modifications such that utilization of a parallel processing capability is possible.

SUMMARY

The DBP concept has been depicted as a promising architectural approach for building higher performance data base processing systems. Intuition indicates, however, that higher performance will not be attainable without expending effort in system research. Research is recommended in areas of DBMS, OS, and system organization to define strategy, functional design, and to evaluate primitives:

Strategy analysis

The fundamental objective of the DBP architecture is higher performance in the processing of data base information. Key to the attainment of this objective is the apportionment of data base processing functions between hardware (the DBP) and software (DBM/OS). Apportionment decisions are most appropriately based upon well defined system strategies formulated knowing the performance impacts of different strategy alternatives. Conventional system strategy and organization can be expected to be modified but intensive examination of the modified strategy is needed in order to make apportionment decisions. Available technology will impact apportionment decisions.

Functional design

Once apportionment decisions have been made, the functional design of a DBP should be investigated. In this design, DBP interfaces, architecture, technology, and primitive functions require specification in concert with the specification of the functional interfaces with the companion central system. Use of primitives and identification of modified central system software and hardware require description.

Environmental primitive evaluation

Primitives, once defined, require evaluation for performance and functional capability. Laboratory implementation of primitives with a modified current system seem relevant to establish practicality and to provide a first approximation to attainable performance.

REFERENCES

1. Peebles, R., and E. Manning, "A Computer Architecture for Large (Distributed) Data Bases," *ACM Proceedings International Conference on Very Large Data Bases,* September 1975.
2. Weissberger, A., "Keeping Pace with a 16-bit Microprocessor," AFIPS 1975, *National Computer Conference.*
3. Freeman, H. A., "More Zip in Your System with Customized Firmware," *EASCON Proceedings,* September 1975.
4. Amelio, G. F., "Charge-Coupled Devices for Memory Applications," AFIPS 1975, *National Computer Conference.*
5. Ypma, J. E., "Bubble Domain Memory Systems," AFIPS 1975, *National Computer Conference.*
6. Moran, D. M., "Memory Multiplexer Data Link—An Intermodular Network," *EASCON Proceedings,* September 1975.
7. Anderson, D. R., "The EPIC-DPS—A Distributed Network Experiment," *EASCON Proceedings,* September 1975.
8. Madnick, S. E., *INFOPLEX—Hierarchical Decomposition of a Large Information Management System Using a Microprocessor Complex,* MIT Report CISR-7, SLOAN WP-770-75, March 3, 1975.

Integrity aspects of a shared data base

by EDUARDO B. FERNANDEZ and RITA C. SUMMERS
IBM Los Angeles Scientific Center
Los Angeles, California

ABSTRACT

A simple model is formulated to represent integrity constraints and to describe the evaluation and enforcement of these constraints in a shared data base environment. The proposed model is applied to define the integrity facilities of a relational data base. This data base consists of a set of base relations, which are presented to the user through application-oriented views. These views are basically joins or projections of the base relations, and the data base is accessed through a language which by referring to predefined views, makes explicit the intentions of application programs. The types of integrity rules incorporated in this system, their description, their evaluation, and the propagation of changes to views into the shared data base, are discussed.

INTRODUCTION

Information protection, i.e., the prevention of illegal disclosure, modification, or destruction, and of invalid modification, of the contents of the data base, is one of the most critical problems of data base systems. For this effect, the data base should include an *authorization system* to prevent illegal access to the information, and an *integrity system* to prevent some types of inconsistencies, introduced by errors of the users or their application programs. By enforcing semantic restrictions on the information, it is possible to insure that the contents of the data base are at least plausible, if not correct, and that no inconsistencies exist between related information.

An integrity system for a data base consists of a set of *assertions* about the contents of the data base (expressed in some suitable language), a *validation mechanism* that checks changes to the data base for compliance with the integrity assertions, and an *enforcement mechanism* that performs some predefined actions upon detecting that one or more of the integrity assertions have been violated. In those systems where users have access to a shared data base through views,[3,17] the integrity system should also contain specifications about how changes are reflected to the shared data.[13]

Typical integrity assertions are statements such as: "Salaries must be positive." "Salaries must be nondecreasing," "Student numbers must be present in enrollment lists." Present commercial systems have only basic capabilities to define and enforce these types of statements,[9] and more advanced integrity checking is left to the application programs. In large shared data bases, it is clear that preserving integrity should be a system function, not left to the individual applications.

A model for the functions of an integrity system is presented in this paper (second section), which describes the structure of the integrity constraints, their evaluation, and their enforcement. This model is applied in the following sections to define an integrity system for a shared relational data base described in previous papers.[11,18] The third section contains a brief description of this data base, the type of integrity assertions included in it, the association of these assertions with data objects, the way of describing the assertions, and their evaluation. The use of compile-time actions to prepare integrity checking, and the effect of changes to the data base through views, are considered. The fourth section compares this approach to other proposals, while a final section provides some conclusions.

A MODEL FOR INTEGRITY

A simple model for integrity is now presented, which can accommodate most of the features found in recent discussions of integrity characteristics.[4,9,10,12,13,17] It is described in terms of a relational model of data,[5] and it is used to guide the selection of integrity features for a relational data base. However, a substantial part of the subsequent development could be applied to other logical models of data as well.

Integrity rules

An *integrity rule* is the 5-tuple (o_l,t_k,p_q,c_n,e_m), where o_l is the *data object* to which the rule applies, t_k is an *access type* which indicates for what type of data base access the rule will be invoked, p_q is an *assertion* stating a semantic constraint which must be true for an occurrence of the object o_l, c_n is a predicate expressing a *condition* which must be true in order for p_q to apply to o_l, and e_m is an *enforcement type* that specifies the action that will be taken by the system if p_q is not true.

For specific systems it may be convenient not to separate the five components of the rules; for example t_k and c_n could be combined to specify a condition for application of p_q, or c_n could be part of p_q.[10,17] Also, in most cases the t_k's are a small and predefined set, and the specification for t_k need not be explicit. However, it is important for conceptual clarity in the design of the integrity system, to separate these components as independent units.

As an illustration let us consider a student data base where record type or relation STUDENT contains the fields NAME, ADDRESS, COURSE, and GRADE. An integrity rule for object STUDENT. GRADE could contain an assertion p_q such as "Grades must be 'A' or 'B' or 'C' or 'D' or 'F'." This assertion would be enforced whenever some access actions involving field GRADE, specified by t_k, are performed; for example if t_k indicates "update, insert," then validation of new values for field GRADE will be performed. However, if a condition c_n is specified, for example: "When COURSE$\neq$'123A'," the validation test will only be applied for those tuples where the COURSE field value is not '123A'. An enforcement type specification could indicate "log and record invalid tuple(s) in INVALID," indicating that on violation of the assertion the name of the program, time of day, etc., will be logged and the tuple or tuples not satisfying the assertion will be stored in a special table called INVALID.

In a system containing many interobject constraints, each of them involving several objects, a more convenient definition for an integrity rule would replace object o_l by a set of objects to which the assertion applies. However, most of the interobject constraints found in practical systems involve only two data objects, such as "Enrolled students must be registered students." In this case, it suffices with splitting the constraint into two constraints of the form indicated earlier, and associating each constraint with each object. In the example, we obtain two assertions: "The set of enrolled students is included in the set of registered students," associated with object "enrolled students," and "The set of registered students includes the set of enrolled students," associated with object "registered students."

In a relational system the objects o_l are either domains or relations. In those data base systems which do not have domains as separate entities,[18] domain constraints are replaced by constraints associated with *field types*, where a field type is a prototype for fields that carries a set of attributes that apply to all the fields based on the corresponding type. The integrity constraints associated with a given type apply similarly to all the fields based on the type. Relation constraints include interrelation between fields or constraints that apply to specific fields in the relation.

There is sometimes a rather fine line between an integrity rule and an access rule. An access rule was described in References 11 and 19 as a tuple (S_i,o_l,t_k,p_q,e_m), where S_i is a subject or requesting entity, and all the other terms are the same as in integrity rules but with a different interpretation. There, o_l is the object accessible to S_i; the type of access authorized to S_l for o_l is given by t_k, and it is only granted if predicated p_q is true. Finally, e_m specifies an action to be performed by the system if an illegal access is attempted. As either access or integrity rules include predicates, application of an integrity rule can depend on the identity of the user, and an access rule can depend on data values. However, access can be decided in some cases by looking only at the names of the requested objects, while integrity control depends always on the contents of the objects involved. Further, the access types t_k specified in an integrity rule always refer to access that modifies the data base, while access control refers to any type of access, including just inspection of the contents of some data unit. Notice also that access rules do not include conditions for their application, but every access must be validated by the system.

Validation of integrity rules

Integrity assertions can be classified into a few basic categories[9,19,13] which correspond to specifications of range, sets of permitted values, format, uniqueness of some value, non-missing values for a field, new vs. old values (transition assertions in Reference 10), and interfield assertions. Validity tests for these categories can be symbolically described as follows.

(a) Update validity test.

Let v be a new value for a field f_i, corresponding to domain or type F_h, which is part of relation R_j. Then, the validity condition is

$$\begin{aligned} &\text{is-valid } (v,f_i,R_j) \rightleftarrows \\ &\qquad \text{for } P=\{p_q \mid (o_l=F_h \vee R_j) \wedge (t_k=\text{`update'}) \\ &\qquad\qquad \wedge (c_n=\text{`true'})\}, \\ &\qquad\qquad P(v)=\text{`true'}. \end{aligned}$$

This test establishes that all the predicates of the integrity rules that apply to object o_l and for which their condition predicates are true must be satisfied, in order for the new tuple including the changed value v to be acceptable. The object o_l is taken to be either the field type for f_i or the relation including f_i, since either of these can contain assertions relevant to f_i.

(b) Insertion/deletion validity test.

Let $(v_1,v_2,\ldots,v_z)$ be a tuple to be inserted into or to be deleted from relation R_j, and that specifies values for fields $f_1, f_2, \ldots f_z$, respectively, corresponding to domains $F_1, F_2, \ldots, F_z$, respectively.

Now the validity condition is

$$\begin{aligned}&\text{is-valid } ((v_1, v_2, \ldots, v_z), (f_1, f_2, \ldots, f_z), R_j) \rightleftharpoons \\ &\quad \text{for } P_i = \{p_q \mid (o_l = F_i \vee R_j) \wedge (t_k = \text{'insert' (delete)}) \\ &\qquad\qquad\qquad\qquad \wedge (c_n = \text{'true'})\} \\ &\qquad\qquad\qquad\quad P_i(v_i) = \text{'true'},\ 1 \leq i \leq k.\end{aligned}$$

These validation tests assume that the assertions are associated with all the relevant fields. For example an assertion "Field f_1 must be a subset of field f_2" would also imply the assertion "f_2 includes f_1".

When users have access to the shared data base through application-oriented views,[3,17,18] it is necessary to have special *reflection rules* which specify how changes to the views are reflected to the shared data base. The resultant tuples, which are tuples of the base relations, must then be validated as above.

A reflection rule has the general structure (v_i, o_l, t_k), where v_i is the view to which the rule applies, o_l is a data object, and t_k an access type (which for fields can be 'null' or 'update', and for relations can be 'null', 'insert', or 'delete'). A rule of this type specifies that for view v_i, a change of type t_k to object o_l in the view results in a corresponding change on o_l in the shared data base.

The following example illustrates the reflection rule mechanism. Let $v_1(f_1, f_2, f_3, f_4, f_5)$ be a view including fields $f_1, \ldots, f_5$, and formed by the join of relations R_1, R_2, and R_3, such that $f_1, f_2 \in R_1$, $f_2, f_3 \in R_2$, $f_3, f_4, f_5 \in R_3$ (that is, f_2 and f_3 are the joining fields). Let the following reflection rules be defined $(v_1, f_1, \text{update})$, $(v_1, f_3, \text{update})$, $(v_1, f_4, \text{update})$, $(v_1, R_1, \text{insert})$, $(v_1, R_3, \text{insert})$. Updates through v_1 are then reflected as follows: update f_1 in R_1, update f_3 in R_2 and R_3, update f_4 in R_3. Integrity constraints referring to the fields of the base relations can now be applied. Insertions through v_1 are reflected as: insert tuple in R_1, insert tuple in R_3 (subject to any integrity constraints affecting R_1 and R_3). Notice that if we had an additional reflection rule such as $(v_1, f_2, \text{update})$, then updates to both f_1 and f_2 are effectively insertions into R_1 and must be reflected as such. Also if the views are formed with projections of some of the relations, then some of the reflected tuples will have unspecified fields, and may thus violate integrity constraints that prescribe specified values for given fields.

FUNCTIONAL SPECIFICATIONS FOR AN INTEGRITY SYSTEM FOR A SHARED DATA BASE

The model of the previous section includes most of the features which are of importance in a practical system. It is used in this section to define an integrity system for a shared data base proposed in previous papers,[11,18] The data description language[19] is used to present the functional capabilities of the system. A mechanism to support these functions is given in the fourth section of this paper.

The data base system

This data base consists of a set of base relations (or *Data Base Structures*), which are presented to the end users through application-oriented views. These views, or *templates*, are basically joins and/or projections of the base relations, constructed for specific purposes. The data base is accessed through an extended high-level language, which, by referring to predefined templates, allows the user programs to manipulate data base elements as any other program variable. The organization of this data base lends itself very well to apply to it an authorization model,[11] which, due to the fact that the extended language makes program data intentions manifest, can be enforced partially at compile time. Even those access decisions which are data-dependent and cannot be enforced at compile time can be prepared at this time, thus decreasing validation overhead at execution time.

The model of data described in Reference 18 uses four kinds of data objects: templates, data base structures, fields, and types. (The term *type* replaces the term *field type* used in earlier papers.) The DB structures constitute the shared system view of the data base, and they are composed of one or more fields. Types, which carry attributes, serve as prototypes for fields. A template is an aggregate (typically a join) of DB structures that is constructed for a specific purpose. A template includes a *defining expression*, which indicates which DB structures participate in its construction and how they are combined. An application program views the data base as a set of templates. The terms DB structure and template will be used in two senses: as structural entities or definitions, and as sets of occurrences of those entities.

Four basic roles for users are contemplated in this system—data base administrators (DBA) define and maintain the shared DB structures and their integrity and authorization rules; data base designers build templates to perform specific functions; application programmers use the predefined templates to write application programs; application users invoke their authorized programs to perform specific actions on the data base.

A data description language for this data base has been proposed in Reference 19, where its functions, use, and syntax are presented in some detail. As its syntax is simple and almost self-explanatory we shall use this language in our examples without further explanations. We are not concerned here with syntactical details and the use of this language is purely to illustrate the functional capabilities of the system.

Integrity rules

Integrity rules can be associated with both types and DB structures. They can be given as part of the object declaration or as independent rules. A rule associated with a type automatically applies to all fields based on that type. The goal is to simplify the descriptive tasks—if a rule is specified for a type, it need not be written for all the fields based on that type. DB-structure rules, enhanced by the relevant type rules, are the basic integrity rules for the shared data base. In addition, the template's defining expression provides criteria for new occurrences created through that template.

An integrity rule can contain references to either built-in functions or procedures provided by an installation. These latter are data base procedures in the sense of the CODASYL DDL report,[4] or formulary procedures,[16] and should be carefully certified. Data base procedures also play a role in the enforcement of integrity and security. Both integrity rules and access rules include an enforcement type, which specifies the name of another system object called an *action*. Some action names are built-in (for example LOG, NOTIFY_SECURITY_OFFICER); other actions are defined by ACTION declarations, which can specify procedure names.

The following are examples of type declarations including integrity assertions:

```
DECLARE EMPNO TYPE WHERE (*>10000 &
*<100000); DECLARE STATUS_CODE TYPE
     FIXED BINARY WHERE (*<100);
```

The declarations for EMPNO and STATUS_CODE specify ranges of admissible values. (The asterisk is used in place of the type or component name.)

The declaration of a DB structure specifies the fields of the structure (by reference to types) and integrity rules. The following are examples of DB-structure declarations:

```
DECLARE 1 SUPPLIER DBSTRUCTURE
  WHERE(NUMBER¬=MISSING),
  2 NUMBER LIKE SUPPLIER_NUMBER,
  2 NAME LIKE BUSINESS_NAME,
  2 STATUS LIKE STATUS_CODE,
  2 CITY;

DECLARE 1 PART DBSTRUCTURE WHERE
(NUMBER¬=MISSING),
  2 NUMBER LIKE PART_NUMBER,
  2 NAME,
  2 COLOR,
  2 WEIGHT;

DECLARE 1 SP DBSTRUCTURE,
  2 SPKEY,
    3 S_NO LIKE SUPPLIER_NUMBER,
    3 P_NO LIKE PART_NUMBER
  2 QUANTITY;
```

SUPPLIER is a DB structure with four fields. The value of SUPPLIER. NUMBER is required; that is, it cannot be undefined (denoted by MISSING in this particular syntax). LIKE indicates correspondence between fields and types, i.e., NUMBER must obey the integrity rules of the type SUPPLIER_NUMBER, CITY of the CITY type, etc. Uniqueness requirements, and integrity rules that involve DB structures other than the object of the rule, are discussed in the fourth section of this paper.

The previous examples expressed integrity assertions as part of the object's declaration. If there is more than one rule for an object, if the rule is complex, or if more dynamic integrity assertions are required, separate statements can be used. The following rule, for example, states that an attempt to set STUDENT NAME to MISSING causes the entire occurrence of STUDENT to be deleted. DELETE is a built-in action.

```
STUDENT IS CONSISTENT WHERE(NAME¬=
        MISSING) ENFORCE(DELETE);
```

The following rule states the requirement of no A's in a specific course:

```
ENROLLMENT IS CONSISTENT WHERE
   (GRADE¬='A') ENFORCE(LOG)
      WHEN(CN='CS123A');
```

The WHEN expression is evaluated first and determines if the WHERE expression is to be evaluated. Unqualified names on the left side of comparisons in the expressions are implicitly qualified by the object name. Reference is to the new value of the occurrence, unless the old value is explicitly specified, as in

```
WHERE(COUNT>OLDVALUE(COUNT))
```

Integrity expressions may be specified in three ways: WHERE, WHERE SOME, and WHERE NONE. The meaning of each form is given in a later section. SOME can be used, for example, to express the requirement that CITY must be an element of CITYVAL, as in

```
CITY IS CONSISTENT WHERE SOME
       (CITY=CITYVAL);
```

To require that PART.NAME be unique, we write,

```
PART IS CONSISTENT
  WHERE NONE(NAME=OLDVALUE
  (NAME));
```

However, this last constraint is awkward, and could be simplified by introducing the concept of uniqueness in the DDL (see for example Reference 10).

Actions

The concept of an *action* provides a generalized way of making events contingent on the state of the data base system. Logging, report generation and various periodic activities are declared as actions. Logging of data accesses is a valuable security tool if the content of the log can be dynamically controlled and if access to the log itself is also controlled. (We refer here to selective logging as opposed to complete journalling that is done for recovery purposes.) Actions also provide a way to start and stop the gathering of statistics or performance measurements.

An action is specified as either (1) a procedure invocation, or (2) combinations of other actions, declared or built-in. It can have initiating conditions specified in the form of a WHEN expression, or can be initiated as enforcement of an access or integrity rule. Built-in actions execute with the rights of the system. Administrator-defined actions execute with the rights of the DBA application and the user class of the definer.[11] The WHEN expression can refer to system data only. For example, to initiate an action whenever a new part is added to the data base

```
DECLARE NEW PART ACTION(LOG,PROC
  (P1))
WHEN (REQUEST.OBJECT='PART' &
  REQUEST.ACCESS='ALLOCATE');
```

In the example above, LOG is a built-in ACTION. P1 is a procedure declared by the administrator, and REQUEST is a system DB structure. The REQUEST and SYSTEM data values are available to the procedure; it can use any other data base items accessible to its application and user class.

There exist some problems associated with the use of actions, not all of which have satisfactory solutions until now. Some of these problems are:

(i) integrity violations within actions, which could result in a never-ending sequence of actions invoking actions;

(ii) name resolution at the moment of invocation of the procedures in the actions;

(iii) access rights of the invoked actions.

A simple, although somewhat unsophisticated solution for (iii) is given in this section. Problem (ii) is discussed later.

Updates through views

From the viewpoint of an application program, all modification of the data base occurs through templates. From the viewpoint of the system as a whole, all modification is effected through changes to DB structures. One of the functions of the view mechanism is to provide the transition from a template-expressed change to an unambiguous DB-structure change. Many problems arise in this context, not all of which have been solved.[6] It is clear, however, that the system's basic integrity rules apply to DB structures. The template definition can place additional constraints on changes made through that template.

A template, like a DB structure, is a structure that can also be viewed as a relation or table. A template can join two or more DB structures, eliminate rows and columns of the resulting table, and permute and rename columns. For example, using the data base definitions, the following template joins SUPPLIER and PART, using SP as intermediate. It also deletes the fields NUMBER and STATUS from SUPPLIER and all fields but NAME from PART, renames CITY to PLACE, and reorders the remaining fields.

```
DECLARE 1 PARTSLOC TEMPLATE
  WHERE (SUPPLIER.NUMBER=SP.S_NO
    & SP.P_NO=PART.NUMBER),
  2 SUPPLIER,
    3 PLACE LIKE CITY,
    3 NAME,
  2 PART,
    3 NAME;
```

The WHERE expression could have been extended by, for example,

```
& SUPPLIER.CITY¬='LONDON'
```

to eliminate rows or occurrences. If a template involves only a single DB structure, and does not select rows, no WHERE expression is needed.

If two fields compared in the WHERE expression are not based on the same type, the template designer receives a warning; if the two cannot legally be compared (according to PL/I rules), the template is rejected. When two fields with unlike attributes are compared, conversion takes place. Attributes may appear on field names in the template; if they do, conversion from or to the attributes of the type will occur when data is accessed.

The template has a *visible* aspect (the portion that appears in the declaration below the defining expression) and a *hidden* aspect (all other fields of the participating DB structures). A program can use only the visible aspect; access and integrity rules can also refer to hidden fields. The user who defines a template must have READ NAME access to all objects whose names appear in the template.[11]

To summarize the definition of a template, it is a projection (not purged of duplicates) and permutation of a subset of the cartesian product of the DB structures named in the defining expression and in the visible aspect. The subset is specified by the defining expression, the projection by the fields appearing in the visible aspect, and the permutation by the order in which the fields appear.

Each template represents a specific intent with respect to data base access. Not all templates are intended to be used for changing the data base, for example. Those that *are* have attributes that limit the kinds of changes that can be made through the template and that resolve any ambiguities about how the change is made.

The UPDATE attribute can appear on template fields. ALLOCATE and FREE are attributes of the template. Allocation of a template by a program, folowing by setting of field value means: "ensure that an occurrence with this value exists in the data base". This could possibly require adding occurrences to more than one DB structure. The ALLOCATE attribute can limit which DB structures will be affected, as in the following attribute for the PARTSLOC template

```
ALLOCATE(SP)
```

which does not allow occurrences of SUPPLIER or PART to be allocated, but only of SP. If there existed no occurrence of SUPPLIER with the NAME value of the new template occurrence, the ALLOCATE would be rejected and an error condition raised. A template with no ALLOCATE attribute cannot be used for allocating. The FREE attribute allows FREE operations to be performed on a template and resolves any ambiguities about which DB-structure occurrences are to be deleted. For example, the program statement

```
FREE PARTSLOC;
```

could be implemented by deleting occurrences of any of three DB structures. The template attribute

```
FREE(SP)
```

specifies which DB structure is intended.

The UPDATE attribute specifies that a template or a template field can be updated, and may also specify propagation of the update to other (possibly hidden) fields of the template. For example, the following template, which is used for changing a part number, updates PART (its source DB structure) and also SP.P_NO, which is compared for equality with PART. NUMBER in the defining expression.

```
DECLARE 1 CHGPN TEMPLATE WHERE
(PART.NUMBER=SP.P_NO),
  2 PART,
    3 NAME,
    3 NUMBER UPDATE(EQUAL);
```

The template's defining expression acts as an integrity rule, since all updates made through the template must satisfy the expression. (Updates need not satisfy the application program selection expression, however).

The UPDATE, ALLOCATE, and FREE attributes specify what can be done through a template, regardless of application or user class, i.e., they implicitly define reflection rules. Access rules further constrain use of the template in specific contexts. A more general mechanism to specify reflection rules could be constructed by the use of actions.

AN INTEGRITY MECHANISM

General aspects

It is clear that a mechanism to perform validation and enforcement of the integrity assertions discussed earlier will involve a good increase in overhead with respect to a system not providing these functions. It is then important to define a mechanism able to support the desired functions at the lowest possible cost. The cost function to be minimized will be the amount of CPU time involved in validating the assertions, and it will be shown that the data base architecture under consideration provides a convenient environment for efficient evaluation of integrity constraints. As it has been pointed out by several authors,[12,14,15] the way in which the data base is structured has a significant effect on the validation effort as well as on the type of inconsistencies that can occur. However, neither the model described here nor this data base have anything special to offer in this respect, and the performance implications of data base structuring will not be considered.

Evaluation and enforcement of integrity rules

As indicated by Florentin,[12] a basic condition to have an effective validation system is a user interface consisting of preformatted transactions. This is the case in this data base, where all manipulations on the data base are made through predefined templates. The prespecified structure of the template makes it possible to validate integrity assertions in a systematic and disciplined way.

Any change to the data base must satisfy both the template's defining expression and all integrity rules for all the DB structures that are changed. The defining expression is first evaluated on the new template occurrence, and the change is rejected if the expression is false.

Integrity rules associated with a DB structure can involve fields in other DB structures. For example, values of one field may be required to be a subset of the values of another. The concept of an *integrity template* is introduced to define how these rules are evaluated when the DB structure is updated. There are three interpretations of the integrity template, corresponding to WHERE, WHERE SOME, and WHERE NONE.[12] Assume a new or changed occurrence of a DB structure DBI, with a new value of dbi. For each integrity rule that has DBI as an object, we consider a WHEN-template whose defining expression is (DBI= dbi & when-expr). If the WHEN-template is empty, the rule is ignored. For each category of WHERE, we consider an integrity template that is the cartesian product of all DB structures appearing in the effective integrity expression (which is the AND of the remaining rules), selected for (DBI=dbi). Then, depending on the category, we require that the effective integrity expression be

(1) true for all occurrences (WHERE)
(2) true for at least one occurrence (SOME), or
(3) true for no occurrence (NONE) of the integrity template.

Changed occurrences become a part of the shared data base when unlocking occurs. During the interval between update and unlocking, violations of certain types of integrity rules may exist, but these are seen only by the process making the changes.

As the data manipulation language used in this system makes explicit the data actions of application programs, it is possible to determine at compile time exactly what the program is intending to do with respect to the data base. This allows to generate code at this time to evaluate and to enforce access rules.[7,11] In a similar way, code can be generated to evaluate integrity rules. In effect, by looking at the type of access of the program with respect to a given data object o_l, it is possible to decide by looking at the t_k of the integrity rules for o_l, which of these rules apply to the given program. The conditions for the relevant rules will normally only be able to be evaluated at execution time since they are usually data dependent, and the same is true for the predicates p_q. However, as the data objects which have to be retrieved to evaluate these predicates are known at compile time, code can be generated for their efficient access. At the same time, code to perform any enforcement action can be generated. Problems with name resolution in the procedures that can be present in p_q, c_n, or e_m, can be decided if compile-time actions include flow of control analysis.[1] Naturally, compiled programs become sensitive to changes in the relevant integrity rules, which result in recompilation of all the programs where they are used.

All this early processing permits considerable reduction in the overhead necessary to perform access control and integrity checking at execution time. For systems where even this overhead is excessive, integrity rules can be evaluated periodically rather than at every modification of the data object. This can be done easily in this system by defining an action to be an integrity check, and giving that action an appropriate WHEN condition. For example,

DECLARE A1 ACTION (INTEGRITY(PART))
WHEN(SYSTEM.TIME='0000'):

As in this system file descriptions are external to application programs, it is possible to use as a mechanism for enforcement a strategy similar to the one proposed in Reference 8, where "surveillance routines" are attached to files where given conditions should be enforced. The only difference is, that instead of these routines being attached by the execution supervisor to the application programs accessing the file, they are attached at compile time.

DBA facilities

A fundamental requirement of an integrity system is to provide a convenient interface for the data base administrator, to permit an easy, consistent, and efficient way of defining and maintaining integrity rules. The data description language presented here in distributed form has been designed to be simple and complete. The integrity rules are themselves stored together with all other data definitions, access rules, action declarations, etc. using the same mechanism as the data files of the shared data base.[19] In other words, integrity rules and all other data descriptions are data base structures and constitute a "data base about the data base," which incorporates the information usually associated with data dictionaries/directories.[20] Some of the integrity rules therefore, could refer to the consistency of the relations storing integrity rules. Also, information about which programs are affected by changes to a given integrity rule should be part of this facility.

RELATED WORK

The separation between the logical and physical aspects of a data base makes it possible to define high-level integrity systems which can be analyzed for consistency and completeness. This has resulted in several approaches to integrity.[2,3,4,9,10,12,13,14,15] It is then important to compare our results to these to put things in perspective.

With respect to integrity, the CODASYL DDL[4] provides the CHECK clause, which can specify either ranges of values, expressed in literals, or the name of a data base procedure. The value range is only one of many types of required integrity constraints; and the procedure invocation mechanism allows integrity requirements to be buried in procedural code, as they are buried in application code in today's data base systems. Such a mechanism has to be provided, but as a last resort after more explicit ways of expressing integrity have been exhausted.

A comprehensive treatment of integrity is given by Stonebraker[17] in describing the INGRES system, in which an integrity assertion is stated as one or more range statements, plus an integrity qualification. The range statements define a cartesian product, and the qualification is true or false for each tuple in that product. This is equivalent to our integrity template, but the lack of a WHEN specification causes rather awkward qualifications. For example, the rule that everyone in the toy department must make more than $8000 is expressed in INGRES as

RANGE OF E IS EMPLOYEE
INTEGRITY E.SALARY>8000 or E.DEPT≠'toy'

as opposed to our

```
EMPLOYEE IS CONSISTENT WHERE
        (SALARY>8000)
   ENFORCE WHEN (DEPT='TOY');
```

Another important relational data base is the SEQUEL system,[2,3] whose integrity aspects are discussed in References 2 and 10. The functional specifications of the integrity subsystem of SEQUEL are in general consistent with the approach presented here. The concept of condition in their integrity rules is imbedded in the assertions. The concept of action is present in their "failure actions," which define how the system responds to integrity violations. Update through views is discussed separately from integrity.[3] The view concept of the SEQUEL system allows the subschema to define joins (as well as other structures) and to convert between units (such as dollars and lire). The template differs from the SEQUEL view primarily in the handling of changes to the data base. A template is the concrete representation of a specific intent regarding use of the data base. Rather than being defined by the casual user of a query language, it is designed by a professional application designer and installed by a DBA.[18] Therefore, rather than applying the "uniqueness rule" of Reference 3 (that a change to a view is permitted only if there is a unique change to the underlying base relations that will result in the view change), we allow the template's definition to choose one of several possible changes, or to disallow a unique change. In any case, it is clear that a system like SEQUEL is compatible with this latter approach, i.e., it would be possible to have data base administrators and application designers building views for casual users.

Florentin[12] has studied integrity from a more theoretical point of view, stressing the value of predicate calculus in defining and evaluating integrity conditions. However, his cost function for the calculation of integrity constraints is based on the number of sequential file searches instead of minimization of CPU time as in our case. Graves has given a very complete discussion of the functions needed in any data description language for integrity purposes.[13] Our concept of condition, and the delayed assertions of Reference 10 are found in his specifications. The syntax of his DDL is COBOL-oriented while our syntax is PL/I-oriented, however most of his concepts are consistent with our approach. He was also the first one considering update effects an integrity problem. Hammer and McLeod[14] have provided a detailed discussion of the nature of integrity constraints; however, they do not consider the problem of updates through views or the evaluation of the constraints. Their work is of great value to define the descriptive aspects of integrity constraints and the functional capabilities of the supporting system.

CONCLUSIONS

A model of the functions of an integrity system for a shared data base has been proposed. Such a model is valuable to guide the design of the functional specifications of integrity systems for specific data bases. The model includes not only the characterization of integrity constraints but also the handling of updates to the data base through views, which is considered here as an important aspect for preserving the semantic integrity of the information stored in the data base. The model puts together different aspects of integrity, which until now have only been partially present in specific proposals.

A relational shared data base presented in earlier papers[18] is then shown to represent a very convenient embodiment of the integrity model. The particular characteristics of this data base, i.e., use of an extended high level language for data manipulation, direct reference to the data base variables, view interface for end users, use of compile-time actions, provide an environment in which the definition, evaluation, and enforcement of integrity constraints can be performed with relatively low overhead, with respect to systems incorporating similar functions.

REFERENCES

1. Allen, F. E., "A Basis for Program Optimization," *Proc. IFIP Congress 71*, North-Holland, Amsterdam, 1972.
2. Boyce, R. F. and D. D. Chamberlin, *Using a Structured English Query Language as a Data Definition Facility*, IBM Research Dept. RJ1318, December 1973.
3. Chamberlin, D. E., F. N. Gray, and I. L. Traiger, "Views, Authorization, and Locking in a Relational Data Base System," *Proc. 1975 National Computer Conference*, AFIPS Press, Montvale, N.J., 1975.
4. *CODASYL Data Description Language*, Journal of Development, Handbook 113, Natl. Bureau of Standards, Washington, January 1974.
5. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM, 13*, 6, June 1970, pp. 377-387.
6. Codd, E. F., "Recent Investigations in Relational Data Base Systems," *Proc. IFIP Congress 74*, North-Holland, 1974.
7. Conway, R. W., W. L. Maxwell, and H. L. Morgan, "On the Implementation of Security Measures in Information Systems," *Comm. ACM*, 15, 4, April 1972, pp. 211-220.
8. Conway, R. W., W. L. Maxwell, and H. L. Morgan, "A Technique for File Surveillance," *Information Processing 74*, North Holland, 1974, pp. 998-992.
9. Date, C. J., *An Introduction to Database Systems*, Addison-Wesley, Reading, Mass., 1975.
10. Eswaran, K. P. and D. D. Chamberlin, "Functional Specifications of a Subsystem for Data Base Integrity," *Proc. ACM-RAND Symposium on Very Large Data Bases*, Boston, 1975, pp. 48-68.
11. Fernandez, E. B., R. C. Summers, and C. D. Coleman, "An Authorization Model for a Shared Data Base," *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 23-31, ACM, New York, 1975.
12. Florentin, J. J., "Consistency Auditing of Data Bases," *Computer J. 17*, 1, February 1974, pp. 52-58.
13. Graves, R. W., "Integrity Control in a Relational Data

Description Language, *Proc. AMC '75 Pacific*, pp. 108-113, ACM, New York, 1975.

14. Hammer, M. M. and D. J. McLeod, "Semantic Integrity in a Relational Data Base System," *Proc. ACM-RAND Int. Symposium on Very Large Data Bases*, Boston, 1975, pp. 25-47.
15. Heath, I. J., "Unacceptable File Operations in a Relational Data Base," *Proc. 1971, ACM SIGFIDET Workshop*, pp. 19-33, ACM, New York, 1971.
16. Hoffman, L. J., "The Formulary Model for Flexible Privacy and Access Controls," *Proc. 1971 FJCC*, AFIPS Press, Montvale, N.J., 1971.
17. Stonebraker, M., "Implementation of Integrity Constraints and Views by Query Modification," *Proc. 1975 ACM-SIGMOD Conference*, May, 1975.
18. Summers, R. C., C. D. Coleman and E. B. Fernandez, "A Programming Language Approach to Secure Data Base Access, *Proc. ACM Pacific 75 Reg. Conf.*, pp. 114-118, ACM, New York, 1975.
19. Summers, R. C. and E. B. Fernandez, *Data Description for a Shared Data Base: Views, Integrity, and Authorization*, Report G 320-2671, IBM Los Angeles Scientific Center, August 1975.
20. Uhrowczik, R. P., "Data Dictionary/Directories," *IBM Systems J.*, *12*, 4, 1973, pp. 332-350.

Designing optimal data structures

by LARRY CLOUGH, WILLIAM D. HASEMAN and YUK HO SO
Carnegie-Mellon University
Pittsburgh, Pennsylvania

ABSTRACT

This paper is concerned with developing a methodology for designing optimal network data base structures. The model uses as an input the logical description of the files and existing network structures which are to compose the new network structure, along with a list of the functional dependencies of each file. From this input, a canonical plex structure is generated which contains all the known data and its relationships in a non-redundant form. This canonical form can be modified by adding additional sets and indexing schemes which can improve the retrieval efficiencies at the cost of additional storage. Given the set of queries or data requests and their relative importance, an operations research model can develop an optimal data structure which minimizes a multiple objective cost function. A methodology is then proposed for loading this optimal structure as well as for detecting when a reorganization or new network structure is necessary.

INTRODUCTION

In the past few years there has been a great deal of research directed toward the development of generalized data base management systems. The major objective of this research was to develop systems which could (1) eliminate redundant data, (2) store all data in one centralized data base, and (3) store not only the data but also the *relationships* between the data. This need to express relationships between data was the impetus for the work which led to the development of the CODASYL DBTG report of 1971.[1-4] Part of this need grew out of the fact that when several files were combined to form a data base and therefore redundant data was removed (i.e., social security numbers existed in more than one file), some mechanisms or relationships needed to be created to represent that information (i.e., something needed to be created to link together those files). In the terminology of the CODASYL report this relationship is called a set. A set consists of an owner-member relationship which is always a one-to-many relationship.

A second research effort which has been concerned with this question of relationships among data are the proponents of the relational approach,[5-11] which was nurtured by E. F. Codd of IBM San Jose Lab. In this approach the data is viewed as a set of tables where all the data contained within a row of that table (tuple) is related. This approach leaves the redundant data in the data base, at least at the logical level, and allows the user to join together various tables of data using the known relationships between the tables. The early work demonstrated that some of the required operators were not valid unless the tables were in a specified or normalized form. The process of converting raw data into the third normal form involves removing repeating groups (such as arrays) as well as removing various functional dependencies. These functional dependencies are indeed some of the relationships between data that the CODASYL report was concerned with representing.

A third research effort in this area has been the work being performed in the file conversion area.[14-36] This work, though preliminary in nature, has been concerned with developing techniques for converting data from one file structure to another. Most of these processes involve defining the input file (source file) and the output file (target file) through the use of a data description language. The programmer then defines the actual conversion process which is then performed to convert the source to the target file.

The objective of this paper is to combine the knowledge gained by the aforementioned research to develop a methodology for automatically designing and loading optimal data base structures. Since the CODASYL approach is currently the accepted generalized data base system, the data structures to be generated will be network or plex structures, which are supported using the set concept. This is not meant to exclude the relational approach in that Bonczek, Haseman and Whinston[12] have demonstrated that a network can be used to store a relational data base and Clough and Haseman[13] have proven that the network approach is an efficient method for storing relational data bases.

It will be assumed that the data to be stored within this optimal data base will consist initially of either a number of networks and/or COBOL files and each will be described using a common data description language. The COBOL file structure is the most common file structure currently being used and most other file structures are a subset of it. It will also be assumed that all functional dependencies are known for each file involved. In order to generate an optimal data structure the list of queries or data requests and their importance must also be known.

OPTIMAL DATA BASE DESIGN

This section of the paper will be concerned with converting the logical description of the input files and networks into a logical description of the optimal network. This entire section will be concerned with only the data description language (logical description) and the data requests (queries) for the resulting data base, and not with the actual data files. This approach of first manipulating the data description language without involving the actual data appears to be a unique idea.

The drawing shown in Figure 1 demonstrates the processes which are discussed in this first section of the paper. The first stage of the design involved (1) developing a data description language (DDL) to describe the input files and their associated functional dependencies and (2) converting those existing networks into hierarchical file structures, so they will be consistent with the COBOL-like files. The DDL which has been developed is an extension to the GPLAN DDL,[3] which is consistent with the DDL proposed by the CODASYL DBTG report. The process involved with converting an already existing network into a group of COBOL-like files is discussed in the third section of this paper.

The primary reason for converting all existing data structures into a hierarchical file is because the second stage can consistently convert each of these files into a normalized form. To first convert a COBOL file into an unnormalized relation the repeating groups must be eliminated and a set of keys for each relationship must be determined using the functional dependencies. The process of converting the unnormalized relations to normalized relationships has been discussed by Codd et al.[5] This is discussed later.

The third stage of the project is concerned with developing a canonical plex (network) structure from the given set of normalized relations. This canonical form contains all the data and all of its known relation-

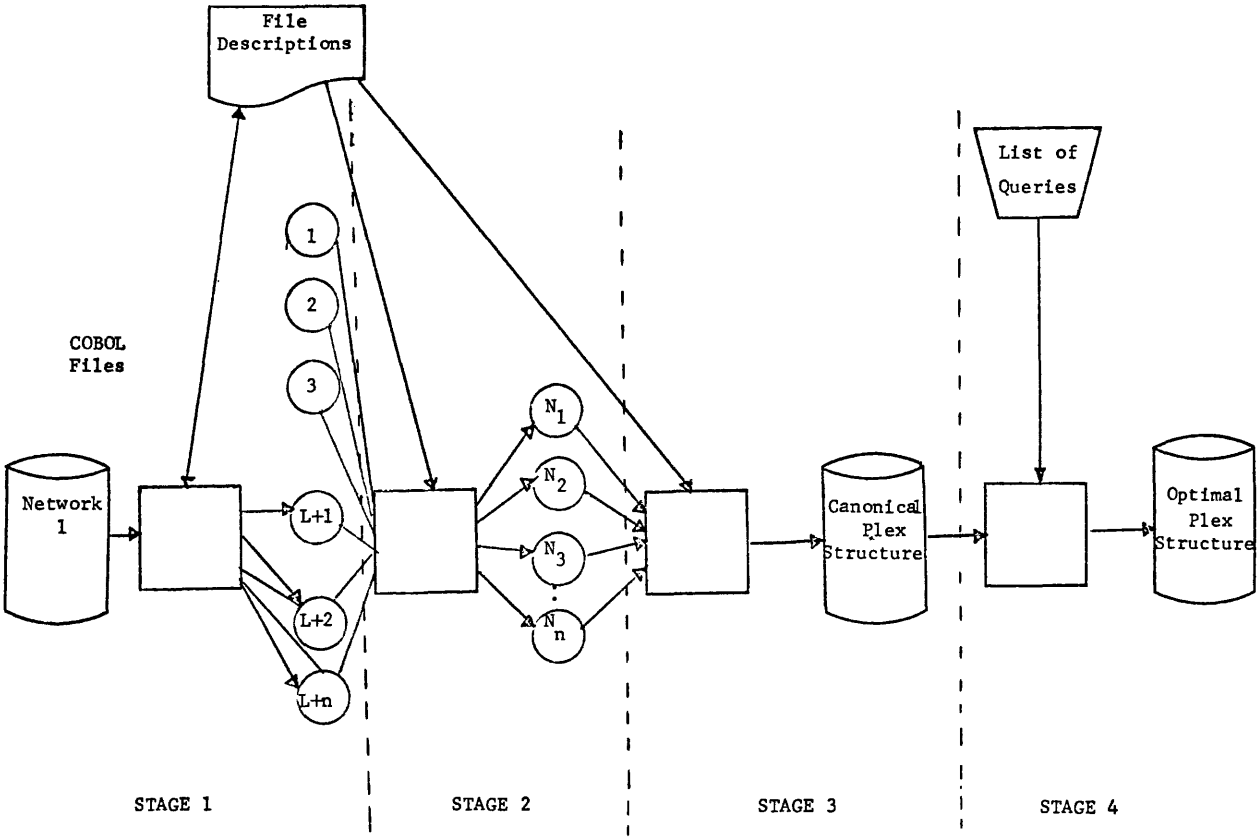


Figure 1—Structure of the data base design

ships. In the context of the relational terminology, this structure can support a relationally complete language.[12] This structure has the network properties of containing no redundant data and no extra set relationships. Since this structure will support any requests for data, this canonical structure could be viewed as being the ideal structure if the types of data requests (queries) are unknown. This structure may also be ideal if the types of data requests are dynamically changing. The discussion in a later section presents some of the operators required to form this canonical structure along with preliminary algorithm for forming the actual structure.

The fourth stage of the project involves trying to optimize this canonical structure in order to answer a given set of queries more efficiently. Various techniques such as adding additional sets or links, creating indexed sets and look-up tables, and splitting apart various relations can possibly increase access efficiencies at the cost of increasing structure costs. The preliminary model presented later looks at how these options can be evaluated in terms of an operations research type of model. This model is concerned with the methodology behind developing criteria for optimal network structures. It is clear that the optimal structure must be an outgrowth of the canonical form to be feasible, however the amount of optimizing must depend on the tradeoffs between access time and mass storage demands.

The following then is a more detailed discussion of the four stages which compose the first phase of the project.

STEP 1—CONVERTING A NETWORK INTO TREES

In order to facilitate the restructuring process, we have chosen the tree representation as the starting data structure for the source files. Since some source files may have network structures, it is necessary to convert them into the corresponding tree structures. In this section we shall investigate preliminarily into this restructuring problem.

The assumed criterion in decomposing the network is to create the least number of trees required to encompass all the data relationships in the network. Thus, the order in which the records are selected as roots plays an important role in the restructuring process. We shall provide in the following the criteria we use in selecting roots in the tree creating process. First, the definitions for the indegree and outdegree are given.

Definition: The indegree of a record x, denoted id(x), is the number of times that x appears as a member of a set relation in the network.

Definition: The outdegree of a record x, denoted od(x), is the number of times that x appears as an owner of a set relation in the network.

The selection criteria

(i) Records with outdegree zero will not be used as roots.
(ii) For all other records, group them according to their indegrees.
(iii) Rank the groups in ascending order of indegrees.
(iv) Rank the records within each group in descending order of outdegree.

An example of the selection criterion is shown in Figure 2.

Notations:

(1) N={(a, b) : record a is the owner and record b the member of a set relation in the network}

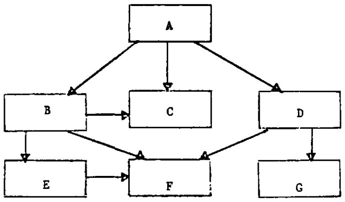
Thus, the data relationships in the network are completely described by N which can be constructed from the DDL description of the network.

(2) R is a set of records rank-ordered with respect to their desirability as roots using the criteria described above.

(3) T_i={(a, b) : (a, b) is a set relation, with owner record a and member records b, in tree i}
TN_i={x : x is a record in tree i}
V_i={x : x is a record to be further expanded in tree i}

The Tree Structuring Algorithm:

```
(1) i=1
(2) If N is empty, then stop.
(3) r=POP (R)*.
    (r is to be used as the root of the next tree to be built)
(4) $TN_i$={r}; $V_i$={r}; T={ }
(5) k=POP($V_i$)
(6) A={(k,y) : (k,y)$\epsilon$N and y$\notin$$TN_i$}
    $T_i$=$T_i$ UA
    (For all y, if y is not already a record in the tree, add all set relations (k,y) in N to $T_i$)
```



	indegree	outdegree
A	0	3
B	1	3
C	2	0
D	1	2
E	1	1
F	3	0
G	1	0

Ranked candidate roots:
(A, B, D, E)

Figure 2—Example of selection criteria

* x = POP(L)
(i) x is the first element in L if L is an ordered list. If L is an unordered set, then x can be any element in L.
(ii) x will be deleted from L.

(7) $N=N-A$
(Delete the set relations, which have been accounted for in the tree structure, from the network data base)
(8) $TN_i=TN_i \cup RANGE\ (A)$**.
(Add the newly attached records to TN_i)
(9) $V_i=V_i \cup \{x:x \in RANGE\ (A) \text{ and } od(x)>0\}$
(For all newly attached records, if their outdegrees are not zero, that is, if they are expandable, add the records to V_i)
(10) If V_i is not empty, then go to (5)
(11) $i=i+1$.
(12) Go to (2).

The example in Figure 3 demonstrates how the network structure shown in Figure 2 would be converted into a group of tree structures using the restructuring algorithm. In this example four COBOL files could be used to demonstrate the designed network structure. It should be reinforced that only the DDL for the new files is being generated, and the actual data is not being converted.

STEP 2—NORMALIZING HIERARCHICAL FILES

C. Delobel[10] has developed a rigorous theory of data in an information system which supports the relational model. He derives the essential aspects of the relational model from the concept of an elementary functional relation between two sets of attributes in an information system. These elementary functional relations define the data structure. All possible relations in the system may be derived from them by use of the join and projection operators. Delobel shows how an elementary set of functional relations may be derived from an arbitrary consistent set of relations. He also addresses the problem of decomposing a collection of data into a set of smaller units which are determined by a set of elementary functional relations and a set of user-supplied constraints upon such decomposition. Delobel presents algorithms for performing these operations.

Bernstein, Swenson, and Tsichritis present a similar approach to defining data structure.[37] They show how a rigorous and simple relational description of data relationships form the functional dependencies. They use a *constructive* approach and show that relations may be synthesized from functional dependencies *algorithmically.*

Functional relations are many-to-one relationships among attributes; owner-coupled sets are one-to-many relationships among records. Can the synthesis of relations from functional relations be used to produce an optimum network structure? A transformation between a relational description of data and an owner-coupled set network description is not simple because of the different modes in which information is stored in the two models of data structure.

(1) Order is used to store information essentially in an owner-coupled-set.

(2) Relationships among data items are stored both in the set structure and in the conjunction of data-items in a record.

(3) The data-item/data-item-value distinction does not capture the domain/attribute value distinction which is present in the relational model.

(4) Owner-coupled-sets are named, and that name contains essential information. Codd has shown that if owner-coupled-sets are treated as analogues of functional relations, spurious information may be created by such associations (see "the connection trap"[6]). For example, there may be more than one owner-coupled-set relationship between two records. A processor which is traversing the network must be sophisticated enough to choose which relationship is appropriate. A relational processor may be more naive because any relationship which it derives will be meaningful if the initial set of functional relations is consistent.

There are three functions which owner-coupled-sets may serve in a network implementation:

(1) elimination of redundant data;

(2) expression of elementary relationships between two records;

(3) improvement of access speed by expression of transitive relationships—i.e., composition of elementary relationships.

Few of the previous objections apply to the conversion of COBOL files into relational form. A COBOL file is very similar to an unnormalized relation. The main difference is that it may have a limit upon the number of occurrences of a repeating group. So it may be transformed almost directly into an unnormalized relation and then normalized. The only significant problem is in determining the primary keys of the unnormalized relations.

Delobel[37] has shown how we may derive a set of elementary functional relations from an initial set, and

** RANGE(M) = {y:M is a set of ordered pairs and (x,y) εM, V x}

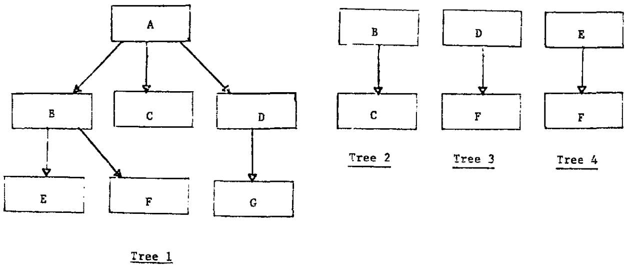


Figure 3

from that elementary set determine a key for a relation. The initial set of functional relationships must either be specified in advance or determined from the set structure of the initial files.

STEP 3—BUILDING A CANONICAL NETWORK FROM RELATIONS

In this section, we shall look into the restructuring problem in consolidating a collection of normalized relations into a network data base. The present stage of the restructuring work is to build a base network structure which will preserve all the data relationships presented in the relations. By creating a network, the data interdependencies across relations are explicitly recognized. This is done by joining the relations on their common domains. Thus, all common domains are represented only once in the network. However, further optimization of the network, such as the splitting of records for efficient accesses or storage savings purposes, will be presented in stage 4.

Three functions used in the restructuring model are described. They are SPLIT, MERGE, and JOIN.

(a) SPLIT

SPLIT is used to extract a specified attribute group from a given relation and to create the linkage (set relation(s)) between the two split parts. First, some notations used will be described.

R: the relation to be split.

D_R: the attribute domain set of R.

E: the attribute group to be extracted from R.

$E' = D_R - E$

$R' = R[E']$ (the projection of R on E').

SPLIT (R, E): the SPLIT function.

Depending on the functional relationship between E and E', three types of splitting might be resulted:

(1) If E uniquely determines E', then create a set relation with owner record E' and member record E.

(2) If E' uniquely determines E, then create a set relation with owner record E and member record E'.

(3) If E and E' assumes a many-to-many relationship, then create a dummy record EE' and two set relations, one with owner record E and member record EE' and one with owner record E' and member record EE'.

(b) MERGE

The function of MERGE is to combine two sets of occurrences of the same record type. The occurrences probably differ in the number and types of set relation pointers. Thus, by MERGING, the occurrences will have a compatible set of pointers.

$MERGE(M_1,M_2)$: the MERGE function where M_1 and M_2 are two sets of record occurrences for the record type M. M_1 is to be merged into M_2.

m: the record occurrence indicator (i.e., without considering the set pointers).

m_1: the record occurrence, with indicator m, in M_1 (m_1 includes all set pointers pertaining to M_1)

m_2: the record occurrence, with indicator m, in M_2.

The process:

(1) for all m, $m \epsilon (M_1 \cap M_2)$, add all pointers in m_1 to m_2.

(2) for all m, $m \epsilon (M_1 - M_2)$, add m_1 to M_2 and attach null pointers to m_1.

(3) for all m, $m \epsilon (M_2 - M_1)$ attach null pointers to m_2 for all owner pointers pertaining to M_1.

An example is shown in Figure 4 of two records being merged.

(c) JOIN

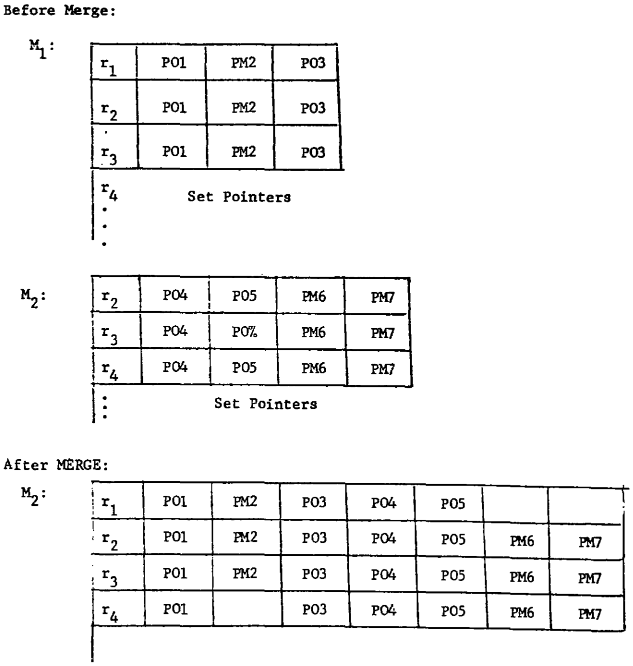
There are two portions of the JOIN operation: (1) to extract a specified attribute group from a given relation; and (2) to merge the extracted group into one already outstanding in the network. Therefore, the function of JOIN is to connect two or more relations on their common attributes.

JOIN (M_1, R): the function

M_1: the attribute group to be extracted and merged into M_2.

R: the relation from which M_1 is extracted.

M_2: the outstanding attribute group with the same type as M_1.



Before Merge:

M_1:

r_1	PO1	PM2	PO3
r_2	PO1	PM2	PO3
r_3	PO1	PM2	PO3

r_4 Set Pointers

M_2:

r_2	PO4	PO5	PM6	PM7
r_3	PO4	PO%	PM6	PM7
r_4	PO4	PO5	PM6	PM7

Set Pointers

After MERGE:

M_2:

r_1	PO1	PM2	PO3	PO4	PO5		
r_2	PO1	PM2	PO3	PO4	PO5	PM6	PM7
r_3	PO1	PM2	PO3	PO4	PO5	PM6	PM7
r_4	PO1		PO3	PO4	PO5	PM6	PM7

Figure 4

The process:

(1) SPLIT (M_1, R).
(2) MERGE (M_1, M_2).

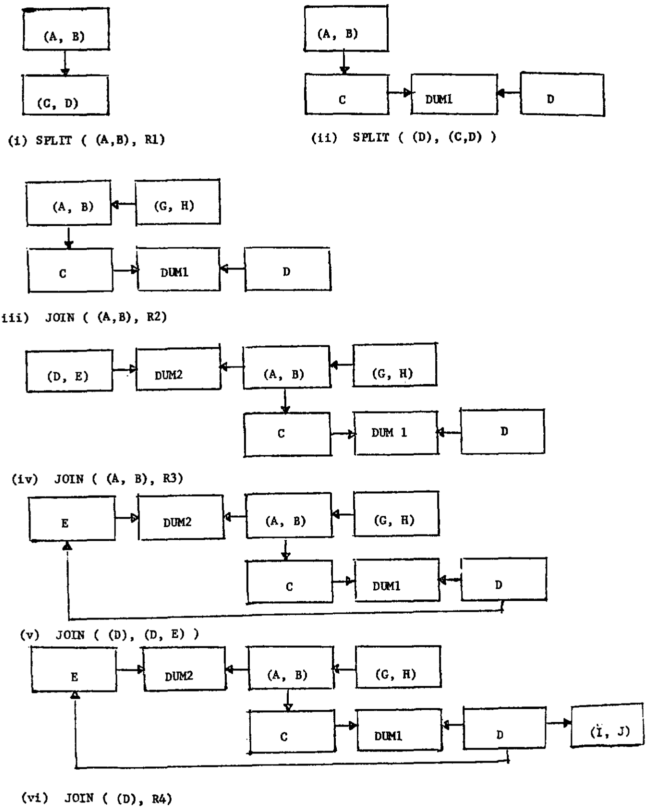
Notation:

C_i: the set of common attribute groups in relation i.
N: the set of attribute groups in the network.

The network generating algorithm:

```
(1)  Number the relations $R_i$, $i=1, 2, \ldots, n$.
(2)  Determine $C_i$ for each $R_i$.
(3)  $N=\{\ \}$.
(4)  $i=1$.
(5)  If $C_i$ is empty, then go to (11)
(6)  Select and then delete an attribute $G_1$ from $C_i$.
(7)  If $G_1=D_{R_i}$ then (i) if $G_1 \epsilon N$,
                                 then MERGE ($G_1$, $G_2$)
                                 else add $G_1$ to N.
                           (ii) go to (11).
(8)  If $G_1 \epsilon N$, then JOIN ($G_1$, $R_i$)
                     else SPLIT ($G_1$, $R_i$).
(8a) $N=(N-\{R_i\})U\{G_1, G_1'\}$
(9)  $R_i=R_i[G_1]$.
(10) Go to (5).
(11) $i=i+1$.
(12) If $i>n$, then stop.
(13) Go to (5).
```

Given the following set of relations and the following assumptions on functional relationships, the network shown in Figure 5 demonstrates the use of this algorithm.



(i) SPLIT ((A,B), R1)

(ii) SPLIT ((D), (C,D))

(iii) JOIN ((A,B), R2)

(iv) JOIN ((A, B), R3)

(v) JOIN ((D), (D, E))

(vi) JOIN ((D), R4)

Figure 5

Relations:

R1(A, B, C, D)	C1={(A,B), D}
R2(A, B, G, H)	C2={(A,B)}
R3(A, B, D, E)	C3={(A,B), D}
R4(D, I, J)	C4={(D)}

Assumptions on functional relationships:

(1) $(A,B) \leftarrow (C,D)$
(2) $C \not\rightarrow D \ \Lambda\ D \not\rightarrow C$
(3) $(A,B) \rightarrow (G,H)$
(4) $(A,B) \not\rightarrow (D,E)\ \Lambda\ (D,E) \not\rightarrow (A,B)$
(5) $E \rightarrow D$
(6) $(I,J) \rightarrow D$

STEP 4—DESIGNING OPTIMAL PLEX STRUCTURES

In order to develop an optimal plex structure it is necessary to investigate what effect the addition of extra sets or access methods will have on the canonical plex structure. In order to evaluate these methods, the demands or queries to be placed on the data structure must be known. Since it is likely that some queries will occur more often than others, it is also nice to know a measure of the frequency with which the query will be asked. For example:

LIST CITY FOR STATE='INDIANA' .03%
LIST NAME FOR CITY= 'INDIANAPOLIS' AND STATE= 'INDIANA' .05%

Once this information is obtained it is possible to identify where the various techniques for decreasing access time may be applied. Initially there appear to be four possible alternatives:

(a) split an existing record
(b) create a new path which bypasses record occurrence
(c) use the CALC option to locate a member directly
(d) use the dynamic pointer to create an inverted structure.

Although other alternatives might exist, this preliminary look will only consider these four techniques.

The format of the queries determines which nodes in the canonical plex structure are to be considered for the optimizing functions. For example, the following query:

LIST POPULATION FOR STATE='INDIANA' AND CITY='LAFAYETTE'.

would suggest that the structure shown in Figure 6 might be a candidate for splitting as well as for using the CALC option. In order to evaluate the increase (or decrease) in core requirements for selecting either or both of these options and the decrease in access time it is necessary to make some assumptions about implementation techniques and size of fields.

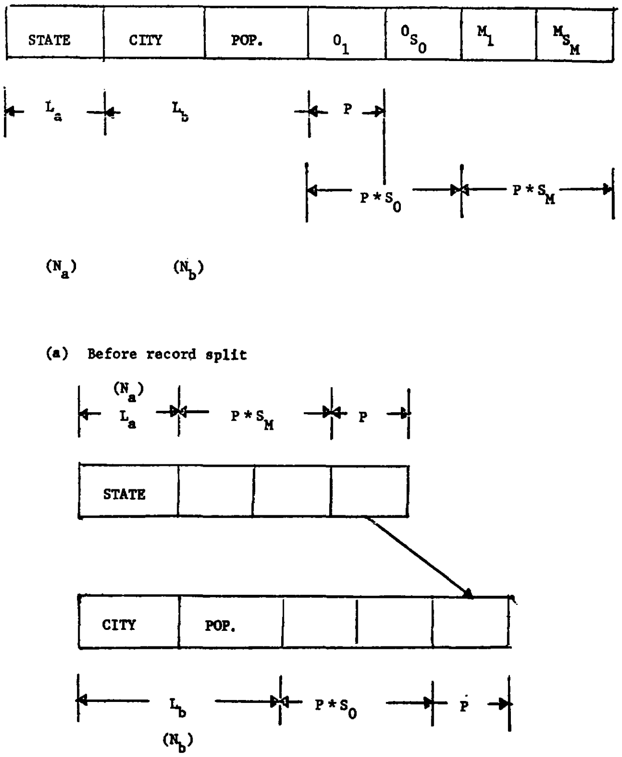
Assumption 1. The canonical plex structure is implemented using a doubly linked list structure, when data and pointers are stored together.

Let:

N_a Number of distinct values for variable STATE
N_b Number of state-city combinations
L_a Length (bytes) of STATE field
L_b Length (bytes) of CITY and POPULATION field
S_0 Number of sets existing record is an owner of.
S_m Number of sets existing record is a member of.
P Length (bytes) for each set of pointers.
c Storage requirements before split.
c′ Storage requirements after split
$\Delta c = c' - c$ Change in storage requirements

The drawing in Figure 6 demonstrates each of these variables. In order to calculate the storage requirements, the following equation can be used to determine core requirements before (c) and after (c′) splitting the record:

$$c = Nb(La + Lb + P^*S_0 + P^*S_m)$$
$$c' = Na(La + P^*(S_m + 1)) + Nb(Lb + P^*(S_0 + 1))$$



(a) Before record split

(b) After record split

Figure 6—Splitting a record

$$\Delta c = c' - c$$
$$\Delta c = La(Na - Nb) + P^*S_m(Na - Nb) + P^*(Na + Nb)$$

where the value of Δc will be the increase (decrease) in number of bytes of storage required to store the record if it is split. It will normally be assumed that this will always be an increase, but for some situations it might possibly be a decrease.

In order to determine the change in access time (Δt) some assumptions will have to be made.

Assumption 2. If the doubly linked list structure is used to locate a single record, it will require looking at one-half the record occurrences on the average.

Assumption 3. The access time T is proportional to the number of records accessed.

Let:

t Records accessed before split
t′ Records accessed after split
$\Delta t = t' - t$ Change in records accessed
$T = wk\Delta t$ Change in access time
w Percent of time specific query requested

Then:

$$t = N_b/2$$
$$t' = N_a/2 + N_b/2N_a$$
$$\Delta t = 1/2(N_a - N_b + N_b/N_a)$$
$$T = wk/2(N_a - N_b + N_b/N_a)$$

This sort of process can be repeated for each of the other possible optimizing alternatives in order to develop the necessary data for the following model.

Let:

$x_i = 1$ if optimization alternative i is selected
0 otherwise for i = 1, 2, . . .
c_{max} = Maximum additional storage available
W_c = Dollar cost per additional bytes of storage
W_t = Dollar cost per additional second of access time
T_{max} = Maximum access time permitted
Δc_i = Change in storage requirements for alternative i
T_i = Change in weighted access time for alternative i

Model:

min $W_c\Delta cx + W_tTx$ minimize total costs

s.t. $\sum_{t=1}^{L} c_ix_i \leq c_{max}$ memory constraints

$\sum_{i=1}^{L} T_ix_i \leq T_{max}$ access time constraints

$x_i \geq 0$
$x_i \leq 1$
x_i integer

where the solution to the problem is those alternatives for which $x_i = 1$ are selected and all others are rejected. The problem of applying multiple options to one record are handled as follows: (See Figure 6)

x_1—split record
x_2—form CALC for STATE
x_3—split and form CALC for State

Using this technique more variables will be generated, however, all intersections can be handled using a linear formulation.

Although this rudimentary model needs further extension, it does propose an approach to solving the question of an optimal network data structure. All of the variables in the model with exception of the weights are included in the proposed data description language which describes the input data files. The selection of the weights, particularly W_t, allows the designer some flexibility to investigate different alternatives. Although the solution to this proposed problem for a large data base may be formidable, it doesn't appear to be totally out of the question.

AUTOMATED DATA LOADING

The second phase of the project involves the development of the methodology for automating and even optimizing the process of loading the data from the specified source files into the optimal plex structure. The initial premise is that the work being done by Ramirez[23] will lead to techniques for generating the code necessary to convert the source file into the target file in a process similar to that shown in Figure 1. Since the input files are initially defined by the data description language and the output of the first phase of the project is a DDL for the target structure, the only missing link will be this code which directs the file conversion. It will be during this second phase that the actual data files will be handled in this data base loading problem.

AUTOMATED RESTRUCTURING

As should be noted in the description of phase 1 of the project, the input files can include existing network structures. The reason for extending the model to include these was to allow for the model to restructure existing data bases. It is clear such an event is necessary when new data items and records are to be added; however, it should also seem clear that if the set of queries or data requests changes significantly, a new set of x_i might be appropriate. The work in this phase will look at extending the model to include the costs of restructuring, so that it can predict at what point the query batch has changed significantly to economically justify reorganizing the logical data structure.

It appears that if no new data is to be added, that the canonical structure will be the same, and only the x_i's will change. It also appears that the loading program might be able to take advantage of this fact and try to minimize the amount of data which has to be physically reloaded. It would appear that this technique would offer a viable solution to the future requirements of a dynamically restructuring data base.

CONCLUSION

The work described in this paper's relevance can be summarized in three areas:

(1) the development of a criterion for "optimal" network structures, using a multidimension objective function;

(2) proposing a methodology of automating the process of designing structures for large data bases;

(3) an attempt to use the work of the CODASYL and relational approaches in a constructive rather than a destructive manner.

As the research on the automation of data loading and restructuring is completed, it is felt that this methodology will lead to great advances in the data base management area.

REFERENCES

1. CODASYL Committee, *Data Base Task Group Report*, Association for Computing Machinery, April, 1975.
2. Haseman, W. D., J. F. Nunamaker and A. B. Whinston, "A Partial Implementation of the CODASYL DBTG Report as an Extension to FORTRAN," *Management Datamatics*, September, 1975.
3. Haseman, W. D., A. Z. Lieberman, A. B. Whinston and J. F. Nunamaker, *Generalized Planning System/Data Management System (GPLAN/DMS) User's Manual* Krannert Graduate School of Industrial Administration, December, 1973.
4. Haseman, W. D. and A. B. Whinston, "A Data Base for Nonprogrammers," *Datamation*, May, 1975, pp. 101-107.
5. Codd, E. F. and C. J. Date, "Recent Investigations in Relational Data Base Systems," *Information Processing '74*, North-Holland, Amsterdam, 1974.
6. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Communication ACM* 13, No. 6, June, 1970, pp. 377-387.
7. Codd, E. F., "Further Normalization of the Data Base Relational Model," in *Current Computer Science Symposia*, Vol. 6, "Data Base Systems," R. Rustin (ed.), Prentice-Hall, Inc., Englewood Cliffs, N.J., 1972.
8. Date, C. J., "Relational Data Base System: A Tutorial," *Information System COINS IV*, J. T. Tan (ed.), Plenum Press, New York and London, 1974.
9 Date, C. J., *An Introduction to Data Base Systems*, Addison Wesley, 1975.
10. Delobel, C., *A Theory about Data in an Information System*, IBM Report RJ964 (#16673), 1972.
11. Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," Process, *ACM SIGFIDET Workshop*, 1971.
12. Bonczek, R., W. D. Haseman and A. B. Whinston, "Structure of a Query Language for a Network Database," 1975.
13. Clough, L. and W. D. Haseman, "Efficiency of Network Structure for Relational Data Bases," 1975.
14. Gasden, J. A., "Report to S3 on Data Definition Languages," SIGFIDET I, 2 (December, 1969).
15. Gosden, J. A., "Software Compatibility: What Was Promised, What We Have, What We Need," *Proc. AFIPS 1968 FJCC*, Vol. 33, AFIPS Press, Montvale, N.J., pp. 81-87.

16. ECMA/TC15/69/15, *First Preliminary Draft Report on the ECMA Data Definition Language.*
17. Fry, J. P., "Introduction to Storage Structure Definition," *ACM SIGFIDET Workshop on Data Description and Access,* 1970.
18. McGee, W. C., "Informal Definition for the Development of a Storage Structure Definition Language," Ibid.
19. Young, J. W., Jr., "A Procedural Approach to File Translation," Ibid.
20. Sibley, E. H. and R. W. Taylor, "Preliminary Discussion of a General Data-to-Storage Structure Mapping Language," Ibid.
21. Taylor, R. W., *Generalized Data Base Management System Data Structures and Their Mapping to Physical Storage,* Ph.D. Dissertation, University of Michigan, 1971.
22. Smith, D. B., *An Approach to Data Description and Conversion,* Ph.D. Dissertation, University of Pennsylvania, 1971.
23. Ramirez, J. A., *Automatic Generation of Data Conversion Programs Using a Data Definition Language (DDL)*, Vols. I and II, University of Pennsylvania, May 1973.
24. Fry, J. P., D. P. Smith and R. W. Taylor, "An Approach to Stored Data Definition and Translation," *ACM SIGFIDET Workshop on Data Description and Access,* 1972.
25. Fry, J. P., R. L. Frank and E. A. Hershey, III, "A Developmental Model for Data Translation," Ibid.
26. Smith, D. P., "A Method for Data Translation Using the Stored Data Definition and Translation Task Group Languages," Ibid.
27. Sibley, E. H. and R. W. Taylor, "A Data Definition and Mapping Language," *CACM,* December 1973, Vol. 16, No. 12.
28. Lum, V. Y., N. C. Shu and B. C. Housel, *Data Translation, Part I: A General Methodology for Data Conversion and Restructuring,* IBM Research Report 1525, July 22, 1975.
29. Housel, B. C., D. P. Smith, N. C. Shu and V. Y. Lum, *Data Translation: Part II: DEFINE—A Nonprocedural Data Description Language for Defining Information Easily,* IBM Research Report 1526, May 6, 1975.
30. Shu, N. C., B. C. Housel and V. Y. Lum, *Data Translation, Part III: CONVERT—A High Level Translation Definition Language for Data Conversion,* IBM Research Report 1515.
31. Ramirez, J. A., N. A. Rin and N. S. Prywes, "Automatic Generation of Data Conversion Programs Using a Data Definition Language," *Proceedings of 1974 SIGMOD Workshop on Data Description, Access and Control,* Ann Arbor, Michigan, May 1-3, 1974.
32. Lin, S. and J. Heller, "A Record Oriented, Grammar Driven Data Translation Model," Ibid.
33. Fry, J. P. and A. G. Merten, "A Data Description Language Approach to File Translation," Ibid.
34. Joint Guide/Share Data Base Requirements Group, *Data Base System Requirements,* November, 1970.
35. Everest, G. C. and E. H. Sibley, "Critique of the Guide/Share DBMS Requirements," SIGFIDET, 1971.
36. Housel, B. C., V. Y. Lum and N. C. Shu, "Architecture to an Interactive Migration Systems (AIMS)," *Proceedings of the 1974 SIGMOD Workshop on Data Description, Access and Control.*
37. Berstein, P. A., J. R. Swenson and Tsichntzis, D. C., *A Unified Approach to Functional Dependencies and Relations,* TR-CSRG-50, February, 1975, University of Toronto.
38. Gerritsen, R., *Understanding Data Structures,* Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1975.

REGIS—A relational information system with graphics and statistics

by J. D. JOYCE and N. N. OLIVER
General Motors Corporation
Warren, Michigan

ABSTRACT

While the relational data management model has been known for some time, it has yet to be proven that such systems can perform efficiently in an industrial environment. This paper describes user experience with and the external highlights of the RElational General Information System (REGIS) which is currently being used within General Motors. This data analysis system combines the features of relational information handling along with graphical, interactive and statistical capabilities. REGIS provides the flexibility of handling unforeseen queries and enables the user to interactively analyze his data by entering commands from terminals. Its use does not require any conventional programming effort. It is possible to interface to user written functions if the need arises.

INTRODUCTION

Although the relational data management approach has been known for some time,[1,2] it has yet to be proven that such systems can perform effectively in an industrial environment. These new concepts have been implemented in the REGIS (formerly RDMS[3,4,5]) relational data management system. This paper describes the highlights of and user experience with this system which is currently being used within the General Motors Corporation.

The **RE**lational **G**eneral **I**nformation **S**ystem (REGIS) is an interactive (conversational) system designed to provide convenient, powerful and flexible information manipulation facilities for information storage, retrieval and analysis. This is an area where long range data management system requirements have been poorly met.[6] Its main power is in its ability to handle unplanned queries and enables the user to utilize intermediate results to determine the future course of his analysis. An emphasis is placed on providing a view of data in which the relationships are easily understandable and easily manipulated to derive new relationships. Of particular importance is the capability to handle groups (sets) of information. This is different from handling one record at a time such as in the CODASYL Data Base Task Group approach.[7] With each REGIS command information may be extracted from a whole set or from the interaction of sets of information.

Capabilities are integrated into the system for those users whose main interests are to use flexible graphical tools to view their data. Other users may want to place an emphasis on statistical analysis of data. REGIS contains elementary statistical programs and planned interfaces to make reasonably smooth transitions between REGIS and advanced statistical packages. The user can prepare his data for analysis utilizing the (relational) data management facilities, analyze it with the statistical operators and plot the results all within one package. Some users have found that the integration of these three broad capabilities in one package is the most useful feature. It is also recognized that other classes of applications will have particular needs for special functions. REGIS provides a simple interface to such functions.

Extensive performance monitoring tools have been installed in REGIS. These will be used primarily by system implementors to measure and assess the effects of new algorithms and any improvement in the existing ones.

SYSTEM OVERVIEW

Figure 1 gives an overview of the major components of the system. The central part of the system consists of the relational information handling capabilities. The graphics part of the system consists of some built-in facilities which will operate on dial-up typewriter or alphanumeric CRT terminals and an integral link to the SIMON graph plotting system. (See the SIMON command description in the Appendix for information about this interactive graphics system.) The SIMON capabilities are invoked with a REGIS command just like any other REGIS facilities. The statistics component includes a number of built-in functions as well as interfaces to other statistical systems. An interface is planned for a general purpose statistical package, a Monte Carlo Simulation system,[8] and a "snowflakes" package. This "snowflakes" (circle diagrams) package is a graphical technique for "browsing" through multivariate data as is indicated in a portion of Reference 9.

REGIS is designed primarily for an interactive mode in which the user can extract and analyze information and make new analyses based on the information which he has just learned. Thus by continuous refinement of the information analyzed, a user can proceed to conclusions by procedures or data dependent methods which might not be evident in the beginning. The user has available a variety of commands which are interpreted as they are entered at a computer terminal. See the Appendix for description of some of the commands, or the Examples section which illustrates their use. The tools of the REGIS system are intended to serve applications where all the queries cannot be defined ahead of time to fully serve an application.

REGIS is available to any user having access to a typewriter or CRT terminal. It runs on an IBM 370/168 computer under the TSS operating system.[10]

USER VIEWS OF DATA AND RELATIONAL SYSTEM

The users view of a relational data handling system is markedly influenced by the approach and terminology by which this subject is introduced to him. Some of the precise terms of relational theory are simply not appropriate to use with potential users. The user must be approached with simple terminology and meaningful explanations of the concepts and ideas involved. It was determined that potential users, including experienced mathematicians, were decidedly turned off by the terms 'ntuple' and 'domain', for example. Replacement of these terms by 'row' and 'column' greatly expedited the communications with new users. This naturally leads into a users view of his data as being made up of simple rectangular tables of rows and columns.

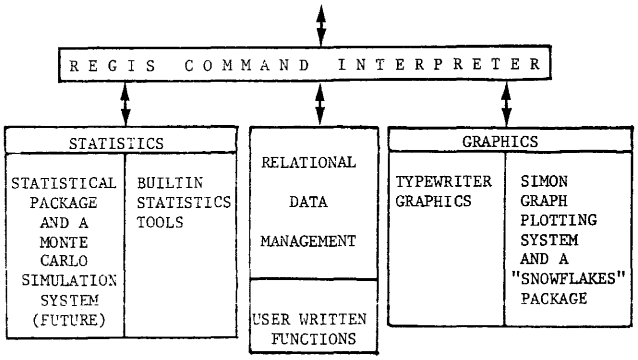

REGIS COMMAND INTERPRETER
STATISTICS
STATISTICAL PACKAGE AND A MONTE CARLO SIMULATION SYSTEM (FUTURE)
BUILTIN STATISTICS TOOLS
RELATIONAL DATA MANAGEMENT
USER WRITTEN FUNCTIONS
GRAPHICS
TYPEWRITER GRAPHICS
SIMON GRAPH PLOTTING SYSTEM AND A "SNOWFLAKES" PACKAGE

Figure 1

NAME	SOCSEC#	SEX	BIRTH
DALEY	371568394	M	40/05/31
SMITH	384246893	F	38/12/23
KINGSTON	354234876	M	45/08/12
WARREN	358123415	M	47/06/12

Figure 2

Data views

The ability of the user to relate views of data to simple rectangular tabular representations provides a common starting point for a wide spectrum of users. Tables were a common denominator for representing data before the advent of computers and continue to be a common representation of data. From these simple concepts, it is relatively easy to lead new users thoughts into understanding that in addition to a table being a repository or collection of data it also represents relationships among the data items in that table. These relationships may be referred to as attributes, or properties of items in the table. Figure 2 shows an example of such a relationship where each person whose name is given in the first column is described by the attributes of social security number, sex and birthdate.

Figure 3 shows another type of relationship which is also contained in a simple table. This example shows a relationship in which the two objects represented by STUDENT and CLASS columns are tied together by the attribute GRADE. The attribute of GRADE does not solely describe either the STUDENT or the CLASS but rather describes the connection (relationship) between the two objects. Other types of relationships could be shown, but the key point, however, is that each relationship can be expressed in a simple tabular form.

After a user has expressed part of his information in a table, it is a natural extension to view his entire data base as a collection of unique groups of data. Additionally, our users view the tables as being related to one another through various common elements of the tables.

STUDENT	GRADE	CLASS
JONES	A	BIOLOGY
SMITH	C	PHYSICS
JONES	B	CHEMISTRY
ADAMS	D	ENGLISH
MARTIN	B	PHYSICS
ADAMS	B	CHEMISTRY
ADAMS	A	PHYSICS

Figure 3

Two observations can be made here from dealing with users in introducing the concepts of relational data bases. The first observation is that users who have been doing computer processing of some of their data have been molded by the computer tools previously available. One impact of this is that users attempt to cram all their relationships into a single table by highly coded representations and replication of data. This has undesirable side effects relating to updating and deletion. Another impact is that significant related data has been ignored in many applications simply because of the lack of facilities to handle multiple relationships. The second observation is that the users find this view of data somewhat simpler than the mechanics of hierarchies or networks. This probably arises from both the backgrounds of the users and the fact that relational implementations generally provide a higher level interface to the data than do the data base packages which require the user to know about the networks and hierarchies.

Relational system views

Once the user has established his external views of the data, his views of the system are much simplified, providing that one can reasonably stay clear of confusing or ominous terminology. The applications to which relational data base techniques are applied tend to be those which the user is examining, condensing, extracting new relationships or insights from groups of data rather than examining or updating individual pieces of data. This acclimates the user to thinking in terms of operations which operate on tables of information. The tables are actually sets of information and the user is frequently applying operators whose foundation is set theory. The user can learn from simple examples about the functions performed by the relational operators. Thus the system appears as a new set of tools to solve his problems quicker and easier. The aura of mystique about relational ideas has been removed and we can move on to the problems of making the systems useful and practical.

APPLICATION EXPERIENCES

Some highlights of feedback from actual applications of a working relational data base manager will be presented here. One application involved personnel data where the information was primarily names, text, codes and dates. One of the areas in this application where the new operators proved to be very useful was in input data validation. By looking at large clumps of data and grouping items together in various ways, many errors either in spelling or in erroneous manual coding of data were isolated. These validation procedures were not carried out before by special programs because of the programming effort required to write and debug computer programs. The relational operations further came into play by making it easy to examine relationships between tables and isolating unusual items for further examination.

It was recognized by the implementors that data handling, whether relational or otherwise, was not an end in itself. Graphical facilities to see data and a reasonable complement of statistics to massage data were an integral part of the design. Feedback from our users indicated that these features which could have been built into many conventional data base packages (and have been in some cases) were sometimes more valuable than the relational power available. The ability to perform aggregate operations (grouping combined with arithmetic functions), joining tables together, subsetting, and selecting items from one table based on criteria from another table have proved to be the most valuable operations to date. Another application consisting mainly of numerical data for an experimental engineering program on noise reduction in new equipment made particularly heavy use of the aggregate functions and the selection of items from one table based on criteria in another table. It is also evident that higher language levels are desirable provided that they can be implemented without significantly increasing the complexity of the language.

EXAMPLES OF DATA QUERIES AND ANALYSIS

Two examples are included to give a flavor of how REGIS is being used at General Motors. (Commands and keywords are shown in capital letters while the set and column names are shown in lower case letters.)

The first example deals with the information in the relationship (or table or set) entitled "failures." This set with its labelled columns is represented as:

```
failures (fail# faildate failtime operator machine#
failtype severity fixdate)
```

The question in the first example is: "Do certain machine operators show a propensity to having particular types of failures when they are operating the machines?"

```
opfail=PROJECTION failures operator failtype
op=AGGREGATE opfail operator COUNT cntop
op=SUBSET op WHERE cntop GE 10
optypes=AGGREGATE opfail operator failtype
  COUNT cntfail
counts=optypes JOIN op operator
COLUMN counts percent FLOAT
counts=counts COMPUTE cntfail * 100 INTO
  percent
counts=counts COMPUTE percent / cntop
result IS SUBSET counts WHERE percent GT
  35.0
```

The PROJECTION command in the first line simply extracts from the source set those columns of interest.

The AGGREGATE command in the second line groups the rows by operator and counts how many rows of failure data are associated with each operator. Now if the user knows that a number of the operators have had only a few failures he might wish to eliminate those operators from further analysis. This is done in the third line with the SUBSET command which selects only those operators who have had 10 or more failures. The next use of the AGGREGATE command in the fourth line groups the rows of failures based on unique combinations of operators and failure types. At the same time this command counts the number of rows in each combination and stores these counts in the column entitled "cntfail." The JOIN command in the fifth line combines the data relationships in the two sets "optypes" and "op." The results of this command are that the operators who had less than 10 failures are eliminated from further consideration and the counts of each operator by failure type and the total number of failures for each operator are stored in adjacent columns. The next three commands (sixth through eighth lines) generate a column for a percent value and compute each failure type as a percent of all the failures for each operator. The SUBSET command in the last line singles out those operators who have an abnormally high percentage of failures of any particular type.

This procedure is one arbitrary solution to a general question. Other solutions could have taken into account absolute numbers of failures by operator and type. Other possibilities include looking at deviations from averages or generating some data which could be plotted to show some deviations from trends.

The next example illustrates use of some of the statistics and graphics procedures. The input data consists of one set with only two columns. A set on quality control data contains the supplier column and the measured hardness of the material in each of a sample of parts in the second column. The set is represented as follows:

qcdata (supplier, hardness)

The desired analysis is stated as follows: "Obtain statistics about the hardness data, then fit a weibull curve and plot a comparison of the raw distribution to a generated weibull curve to give a visual indication of the closeness of fit."

```
stats=STATISTICS qcdata
tempset=PROJECTION qcdata hardness SORT
wb=WEIBFIT tempset
wbcurve=CURVE wb
DEFAULT INTERVALS 12
rawdata=DISTRIBUTION qcdata hardness 100
  BY 5
comb=wbcurve COMBINE rawdata
SIMON comb
GRAPH frequency VS hardness
ADDGRAPH yhardness VS xhardness
```

The STATISTICS command in the first line stores in the "stats" set statistics about all the numeric columns in the "qcdata" set (in this case only the hardness column). The PROJECTION command in the second line sorts the hardness data in preparation for the weibull fit command which follows. After the weibull parameters are generated (third line) a curve containing 100 points will be generated with the CURVE command in the fourth line. The next two commands set up the parameters for and generate a distribution of the raw data. The COMBINE command (seventh line) merely puts the data for the two curves in one set in preparation for the graph plotting command which must have all its data in one set. The SIMON command brings the user into a graph plotting system which permits the use of any of the SIMON commands to overlay curves, label axes, add titles, add legends, etc. The GRAPH and ADDGRAPH commands in the last two lines of the example illustrate the commands necessary to plot two curves on the same graph.

For more information about the command language see the Appendix.

PERFORMANCE MONITORING

REGIS contains automatic performance monitoring facilities which continuously record various phases of its performance for the usual purposes of locating inefficiencies and predicting performance (cost and response time) as a function of command sequences and data base sizes. In addition, command sequences are monitored to determine whether certain commands should be restructured in simpler or more powerful relational commands for the user. Since there is a wide variety of applications and users with diverse backgrounds, the data concerning performance and distribution and usage of commands in user environments will also be useful to designers of data base hardware and firmware which incorporate relational operations. The monitoring information collected is stored in normal REGIS sets and thus can be later analyzed utilizing this data analysis package.

CONCLUSIONS

The development of a relational data base implementation has resulted in a worthwhile tool for a wide variety of applications. The new relational concepts are practical in a problem solving environment for industrial applications. Concern over efficiency of such a generalized approach is still justified. However, use of the system does eliminate the traditional specialized programming effort. We feel that this savings has greatly offset any execution inefficiencies in this general purpose package. It is also clear from the performance monitoring data that substantial improvements in efficiency can be made. Improved sorting and searching capabilities will make such systems economically applicable to even wider classes of applications.

Simple views of the data to be handled and the commands for manipulating the data have been very attractive to users. A manual explaining the concepts in simple terms and augmented with a few examples has been sufficient for a number of users to make effective use of the system without any training or consultation. The ability to carry out unplanned queries and data reduction interactively (either via local or remote dialup terminals) has been valuable to users. A type of macro facility in which a knowledgeable user prepackages command sequences for other less knowledgeable users has been useful in those applications where sequences are often repeated. User acceptance of the system was greatly encouraged by the provision of a comprehensive variety of graphical and statistical analysis capabilities as supplements to the relational data handling facilities.

APPENDIX—COMMAND LANGUAGE

The REGIS command language is a non-procedural[11] relational algebraic language believed to encompass the most useful relational functions. Commands have the following general format:

$$\text{[set } \left\{\begin{matrix}\text{IS}\\ =\end{matrix}\right\} \text{ [set]] KEYWORD [parameter...]}$$

(Notation explanation: Uppercase words denote a keyword that must appear as shown; items contained within square brackets [ ] are optional; items followed by the ellipses ... may be used once or more; multiple items stacked and enclosed in braces { } represent alternatives for a choice of one item.)

This command language operates in any one of two modes depending on the leftmost word of the command: Either it is a command name in which case it is usually a control command and the results are displayed at the terminal, or it is a set name in which case a new (or old) set is used to store the results of the specific operation. Varying number of options (if any) can be specified in the parameter field of each command.

Examples:

COMMANDS FROM set
LIST set
set=SUBSET set
set IS set1 UNION set2

The syntax and brief description of the most useful commands follows. For additional information see Reference 12.

COLUMN set column [type]

The COLUMN command adds a new column with the specified name to the specified set.

ADDROW set rowvalues ... $END

This command adds row(s) to the specified set.

$$\text{COMMANDS } \left\{\begin{matrix}\text{TO}\\ \text{FROM}\end{matrix}\right\} \text{ file}$$

COMMANDS enables the user to store and/or execute more than one command at a time. It is a convenient facility through which a long sequence of commands can be executed by entering only one command.

SPECIAL name setin setout

The SPECIAL command allows the user to execute any program which he has previously created. Thus the user is able to add to the REGIS package extra capabilities tailored to his specific needs. (For example: a report generator.)

$$\text{BARCHART set column} \left[\text{minimum} \left[\left\{\begin{matrix}\text{TO maximum}\\ \text{BY increment}\end{matrix}\right\}\right]\right]$$

BARCHART produces a horizontal bar chart of the values in a specified range of one column of a set.

SIMON set

This command invokes the SIMON PLOTTING SYSTEM to obtain graphs of REGIS data. This simple graphics system, developed at General Motors Research Laboratories, enables the user to produce good formatted graphs of data stored in any REGIS set. The user can restyle the graph via SIMON interactive commands. The graph is generated on the TEKTRONIX storage tube terminal while hardcopies can be generated on CALCOMP plotters or SC4060 film recorders.

REVISE set column [#[TO #]] WITH newvalue [increment]

REVISE enables replacing a specified range of column values in a set with a constant and optional increment.

resultset=PROJECTION set columns ... [ETC]
[SORT [WEED] [DESCENDING]]

The specified columns of the source set are copied to the resultset. If the SORT option is specified then the resultset is sorted on these columns.

resultset=SUBSET set

$$\left[\left\{\begin{array}{l}\text{\# [TO \#]}\\ \text{WHERE Compare} \left\{\begin{matrix}\text{AND}\\ \text{OR}\end{matrix}\right\} \text{Compare] ...}\\ \text{SAMPLE samplevalue}\end{array}\right\}\right]$$

The Compare syntax is: columna Operator $\left\{\begin{matrix}\text{columnb}\\ \text{value}\end{matrix}\right\}$

Operator can be any of the following: =or EQ; ≥=or NE; >or GT; >=or GE; <or LT; <=or LE. ¬

SUBSET places rows of set into resultset. The WHERE clause compares values from columna to either values in columnb or a constant value. The comparison conditions can be nested in multiple levels.

$$\text{rset=set COMPUTE} \begin{Bmatrix}\text{columna}\\ \text{value}\end{Bmatrix} \text{Operator} \begin{Bmatrix}\text{columnb}\\ \text{value}\end{Bmatrix}$$
[INTO columnc]

Operator can be any of the logical operators described with the SUBSET command and also any of the following: + (plus) : − (minus) ;* (multiply) ; /(divide) ; EXP; REM or TRUNC.

This command performs the arithmetic operation on all elements of the specified columns. The arithmetic operation between each value of columna and the corresponding value of columnb, or constant values, is performed and the result is placed in columnc.

resultset=AGGREGATE set columns ... [COUNT column] [SUM columns ...] [AVERAGE columns ...] [MINIMUM columns ...] [MAXIMUM columns ...]

The source set is subdivided into groups based on distinct combinations of values for the specified columns. One row for each group is placed in the resultset. A count, sum, average, minimum and maximum of the consolidated columns can be optionally obtained.

resultset=STATISTICS set [column ...]

This command calculates values such as: minimum, average, maximum, sum, standard deviation etc.

$$\text{rset=set1} \left\{\begin{matrix}\text{UNION}\\ \text{INTERSECTION}\\ \text{XOR}\\ \text{DIFFERENCE}\\ \text{JOIN}\\ \text{COMPOSITION}\end{matrix}\right\} \begin{matrix}\text{set2 [cols1...}\\ \text{[MATCH cols2...]]}\end{matrix}$$

The UNION and XOR commands place in rset rows from set1 and set2; UNION places all rows of set1 and all rows of set2 which are different in the matching columns into rset; XOR places only those rows of either set which are not in both sets into rset. Rows are compared on the basis of named columns from set1 matched up with the named columns from set2. The INTERSECTION (DIFFERENCE) commands place in rset rows from set1 when the specified columns of the row in set1 match (do not match) those of set2. The JOIN and COMPOSITION commands place in rset combinations of rows whose specified columns match. JOIN places all columns of set1 and set2 in rset. COMPOSITION, on the other hand, excludes the matched columns.

$$\begin{matrix}\text{resultset=SELECTION set1 [FOR column1a]}\\ \text{WHERE column 1b} \begin{Bmatrix}\text{EQANYOF}\\ \text{EQALLOF}\end{Bmatrix} \begin{matrix}\text{column2}\\ \text{IN set2}\end{matrix}\end{matrix}$$

For the EQANYOF option, this command places in resultset those rows of set1 for which the value of column1b is equal to any value of column2 in set2. The EQALLOF option partitions the rows of set1 into groups, one group for each distinct value in column1a. If the values of column1b for a group include all the different values of column2 in set2, then that entire group of rows is placed in resultset.

ACKNOWLEDGMENTS

We wish to acknowledge the contributions of Kevin Whitney to the original experimentation, design and implementation of the system. Others who contributed to the design and implementation include Karen Nielsen, Steve Holland and Dick Rawson. We would also like to thank G. G. Dodd for his support of this project and advice during the course of its development.

REFERENCES

1. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Communication of the ACM*, XIII, 6, June 1970, p. 377-387.
2. ———, "Normalized Data Base Structure: A Brief Tutorial," *Proceedings of the 1971 ACM SIGFIDET Workshop*, pp. 1-18.
3. Whitney, VK. M., "A Relational Data Management System (RDMS)," *Proceedings of the COINS-72*, December, 1972.
4. ———, "Fourth Generation Data Management Systems," *Proceedings of the National Computer Conference*, 1973.
5. ———, "Relational Data Management Implementation Techniques," *ACM SIGFIDET Workshop on Data Description, Access and Control*, Ann Arbor, Michigan May 1-3, 1974.
6. Joyce, J. D., J. T. Murray and M. R. Ward, "Data Management System Requirements," *Data Base Management Systems*, edited by D. A. Jardine, North-Holland Publication Co., pp. 115-128.
7. *CODASYL Data Base Task Group Report*, ACM, New York, NY, April, 1971.
8. "Search," General Motors Research Laboratories, *A New Dimension in Dimensional Tolerancing*, Vol. 9, 6, December 1974.
9. McDonald, G. C. and J. A. Ayers, *Some Applications of the "Chernoff Faces": A Technique for Graphically Representing Multivariate Data*, Research Publication GMR-1555, General Motors Research Laboratories, January 1974.
10. IBM System/360, Time Sharing System, *Command System User's Guide*, Order No. GC282001.
11. McLeod, D. and M. Meldman, "RISS—A Generalized Minicomputer Relational Data Base Management System," *Proceedings of the National Computer Conference*, 1975.
12. Joyce, J. D. and N. A. Oliver, *Preliminary Users Manual for REGIS Information System*, Research Publication GMR-2008, General Motors Research Laboratories, Warren, Michigan.

Query-by-example—Operations on hierarchical data bases

by MOSHÉ M. ZLOOF
IBM Thomas J. Watson Research Center
Yorktown Heights, New York

ABSTRACT

Query-by-Example is a high level non-procedural data base language which provides the end user with a simplified unified interface for *querying, updating, defining,* and *maintaining,* the data base, as well as imbedding various *integrity* and *authority* constraints. When querying the data base the user fills in, through a keyboard display, an example of a possible answer in a skeleton of the logical structure of the data base. As demonstrated in previous work, when the data base is relational, skeleton tables are used. In this paper, we show that the Query-by-Example operations are in fact independent of the structure of the data base. In particular, we demonstrate that if the view at the user interface level is hierarchical, the query is again accomplished by filling-in an example of a possible answer, but, in this case, a skeleton of the hierarchy is utilized. It is, also, shown how the user can map a relational view into a hierarchical view and vice versa, formulating queries that involve both views simultaneously. Finally, a linear version of Query-by-Example is made available for situations where a display facility is unavailable.

INTRODUCTION

Query-by-Example is a high level non-procedural data base language which provides the end user with a simplified unified interface for *querying, updating, defining,* and *maintaining* the data-base, as well as embedding various *integrity* and *authority* constraints. When querying the data base the user fills in, through a keyboard display, an example of a possible answer in a skeleton of the logical structure of the data base. Our main philosophy behind Query-by-Example was to allow the user to learn very little to get started, and keep the number of objects and concepts that has to subsequently be learned, to cover the whole language, at a minimum. And in fact, the results of various behavioral tests conducted in our labs to teach non-programmers the Query-by-Example language[5] showed that in less than three hours the users could ask quite complicated queries as powerful as first order predicate calculus.

Initially, Query-by-Example was started as a query language for relational model of the data base[1,2] as was introduced by E. F. Codd[10,12] but since then it was extended to a whole data base language with facilities to update, define, and maintain the data-base;[3,14] however, to keep the simplicity of the language intact, most of the operations used for querying the data base are kept in use for updating, defining, and maintaining that data base, thus providing the user with only *one* unified interface. For that reason we did not feel it is necessary to change the name of the language for the other modes of operations, such as Define-by-Example or Update-by-Example etc., since the same Query-by-Example operations are also used for these modes.

In this paper we introduce Query-by-Example operations on a hierarchical view of the data base (view being what the user sees at the user interface level, independently of its underlying model—for example, a hierarchical view can be built on top of a relational model of the data). Here a query is again formulated by filling in an example of a possible answer through a keyboard display, but in this case a skeleton of the hierarchical structure is utilized. It is also shown how the user can map a hierarchical view of the data into a relational view and vice versa. This implies that the Query-by-Example operations are independent of the view of the data base be it relational, hierarchical, network, or any combination of the three.

The reason why the user should be given the options to view the data base in the above three different ways are as follows:

—It is sometimes more intuitive for the user to view part of his data base as a hierarchical tree (such as an organizational chart), although the underlying model may be relational.

—Although we are of the opinion, as are many others, that the relational approach which was proposed by Codd, will in the future be implemented on large-scale systems, the current large-scale implementations, however, are primarily hierarchical or network.

Therefore, given a currently standing hierarchical data base model (such as IMS) with a programmer who is already trained to work with such a hierarchical structure, it would be advantageous to be able to replace current procedural data languages (such as DL/1 or GIS) by a high level non-procedural language (such as Query-by-Example) that operates on the *same* hierarchical views of the data base.

While the procedural language programmer *must* have knowledge of the hierarchical structure and access paths of the data base in order to 'navigate' through the tree paths, the Query-by-Example user is only required to have knowledge of the data base structure. The transformation of a non-procedural Query-by-Example statement into a procedure in a lower level language is done by the system through a proper interpreter.

The last reason is that of enhancing performance. If in a given system performance is a major factor, one can enhance it by letting the user view the data base as a hierarchy. Once he is aware that certain paths are more efficient than others, the user may have some control over the performance of the system even though the language is still non-procedural.

A summary of the basic concepts of Query-by-Example as they apply to a relational view of the data base is presented in the second section of this paper. In addition, a linear version is introduced for situations where a display facility is unavailable. In the third section, we introduce Query-by-Example operations on a hierarchical view of the data base. Section four is a combination of the basic concepts of the second and third sections, such that the user can formulate a query concerning both relational and hierarchical views simultaneously. It is also shown how the user can map one view into another. In the fifth section, we comment on how the user can participate in enhancing the performance of the system.

SUMMARY OF THE MAIN FEATURES OF QUERY-BY-EXAMPLE

When dealing with a relational data base, the user basically formulates his query by *filling in* skeleton tables with an example of a possible answer through a keyboard display. In fact, to get started on the system, the user need only distinguish between the following two entities:

1. The 'example element' (variable) which must be underlined and
2. The 'constant element' which should not be underlined.

In addition, the function denoted by 'P.' stands for 'print': the user inserts 'P.' before any data he wishes to be outputted.

A relation is basically a table, and an example of such a table is the employee relation given in Figure 1.

EMP	NAME	SALARY	MANAGER	DEPARTMENT
	ANDERSON	8K	SMITH	TOY
	MORGAN	10K	LEE	COSMETICS

Figure 1—Employee relation

As in Reference 1, the concepts of Query-by-Example are introduced through illustrations of queries and their answers. The queries are drawn from the following relations which are a part of a department store data base.

—EMP (NAME, SAL, MGR, DEPT)

—SALES (DEPT, ITEM)

—SUPPLY (SUPPLIER, ITEM)

—TYPE (ITEM, COLOR, SIZE)

The EMP Table specifies the name, salary, manager, and department of each employee.

The SALES Table is a listing of the items sold by departments.

The SUPPLY Table is a listing of the items supplied by suppliers.

The TYPE Table describes each item by color and size. A sample of the above data base is shown in Figure 2. At this point we are assuming that these tables are made available to the user upon calling them by name.

Actually, in a current implemented prototype,[13] the sequence of user operations in constructing a query is as follows:

EMP	NAME	SALARY	MGR	DEPT
	JONES	8K	SMITH	HOUSEHOLE
	ANDERSON	6K	MURPHY	TOY
	MORGAN	10K	LEE	COSMETICS
	LEWIS	12K	LONG	STATIONARY
	NELSON	6K	MURPHY	TOY
	HOFFMAN	16K	MORGAN	COSMETICS
	LONG	7K	MORGAN	COSMETICS
	MURPHY	8K	SMITH	HOUSEHOLD
	SMITH	12K	HOFFMAN	STATIONARY
	HENRY	9K	SMITH	TOY

SALES	DEPARTMENT	ITEM
	STATIONARY	DISH
	HOUSEHOLD	PEN
	STATIONARY	PENCIL
	COSMETICS	LIPSTICK
	TOY	PEN
	TOY	PENCIL
	TOY	INK
	COSMETICS	PERFUME
	STATIONARY	INK
	HOUSEHOLD	DISH
	STATIONARY	PEN
	HARDWARE	INK

SUPPLY	ITEM	SUPPLIER
	PEN	PARKER
	PENCIL	BIC
	INK	PARKER
	PERFUME	REVLON
	INK	BIC
	DISH	DUPONT
	LIPSTICK	REVLON
	DISH	BIC
	PEN	REVLON
	PENCIL	PARKER

TYPE	ITEM	COLOR	SIZE
	DISH	WHITE	M
	LIPSTICK	RED	L
	PERFUME	WHITE	L
	PEN	GREEN	S
	PENCIL	BLUE	M
	INK	GREEN	L
	INK	BLUE	S
	PENCIL	RED	L
	PENCIL	BLUE	L

Figure 2—Sample data base

The user is initially presented with a display with a blank skeleton table in which he enters through a keyboard, the appropriate table name in the table name field. Upon pressing a function key, the column names are automatically filled in by the system. If needed, a different function key will create additional blank skeleton tables etc. After having on the screen all the required skeletons with their column names, the user proceeds to fill in these tables with elements to satisfy the stipulation of the query. For more details, please see Reference 13.

Let us now proceed with query examples.

Examples:

Q1. Print the red items.

The user fills in the TYPE Table in the following manner.

TYPE	ITEM	COLOR	SIZE
	P.PEN	RED	

Explanations:

Since the query is concerned with red items, RED is a constant element and is, therefore, not underlined. On the other hand, the underlined element PEN is the example element and is entered as an example of a possible answer. Actually, a pen may not necessarily be an element of the data base and can be substituted by DRESS, WATER, or a variable X without altering the meaning of the query. One of the reasons we are using an example element is that it gives us the freedom to use an entity which is partially variable and partially constant (such as PEN, meaning: Print the red items that start with the letter P). Also, for the above simple query, one can dispose of the example element PEN altogether, entering only 'P.'. The SIZE Column can be left blank or filled in with an example element.

Considering the sample data base, the answer to Q1. is:

ITEM
LIPSTICK PENCIL

For cases where a display is not available, a linear version of Query-by-Example, which is a straight forward mapping of the tabular version, is also available, and Q1 in linear form will be written as follows:

Q1. TYPE (ITEM :P.PEN,COLOR :RED)

Q2. Find the department(s) that sell(s) an item(s) supplied by the supplier Parker.

Here the user fills in both the SALES and the SUPPLY Tables as follows.

SALES	DEPT	ITEM
	P.TOY	PEN

SUPPLY	ITEM	SUPPLIER
	PEN	PARKER

ANS:

DEPT
HOUSEHOLD TOY STATIONARY HARDWARE

Explanation:

The example element PEN (linking variable) is included in both tables, implying if an item is sold by the department in question, that same item has to be supplied by Parker.

In linear form Q2 will be written as follows:

Q2. SALES (DEPT :P.TOY,ITEM :PEN)
SUPPLY (ITEM :PEN,SUPPLIER :PARKER)

If the user wishes to retrieve information from two or more tables in the form of a new relation (table), he first sets up a skeleton of the required table by specifying its attributes, and then fills in this skeleton table with linking elements. The assignment of a name to the new relation being outputted is optional. This is illustrated in the next query.

Q3. List the departments and their corresponding suppliers.

SALES	DEPT	ITEM
	TOY	PEN

SUPPLY	ITEM	SUPPLIER
	PEN	BIC

DEPT	SUPPLIER
P.TOY	P.BIC

Explanation:

Here the user groups the two columns DEPT and SUPPLIER to form a skeleton of a table (which may or may not be identified by name). The example element PEN appearing in both the SALES and the SUPPLY tables indicates a natural join on the common domain ITEM.

Q3 in linear form:

Q3. SALES (DEPT :TOY,ITEM :PEN)
SUPPLY (ITEM :PEN,SUPPLIER :BIC)
(DEPT :P.TOY,SUPPLIER :P.BIC)

If the user wishes to save an intermediate relation, which he created from columns of existing ones, for use in subsequent queries, he must assign a name to this relation and precede it by the command 'SAVE.'.

Q4. Create an intermediate table DEPT-SUPP such that it contains all the departments and their corresponding suppliers.

SALES	DEPT	ITEM
	TOY	PEN

SUPPLY	ITEM	SUPPLIER
	PEN	BIC

SAVE. DEPT-SUPP.	DEPT	SUPPLIER
	TOY	BIC

Explanation:

The 'SAVE.' command will save this intermediate relation DEPT-SUPP. The user can now call this table by name and query it in the usual manner. A 'P.' in front of TOY & BIC would cause the system to print out the data (in addition to saving it), which would be the same as the table outputted in Q3. A second command, 'REMOVE.', is used in the same manner as 'SAVE.' to dispose of intermediate relations when no longer required.*

Q4 in linear version:

Q4. SALES(DEPT:TOY,ITEM:PEN)
SUPPLY(ITEM:PEN,SUPPLIER:BIC)
SAVE.DEPT-SUPP(DEPT:TOY,SUPPLIER:BIC)

Q5. Find the name(s) of any employee(s) who earns more than his (their) manager(s).

EMP	NAME	SAL	MGR	DEPT
	P.JONES	>10K	HARRIS	
	HARRIS	10K		

ANS:

NAME
LEWIS HOFFMAN

Explanation:

If HARRIS is an example of such a manager and if HARRIS earns 10K (as an example) then JONES is an example of an employee who earns more than 10K (indicated by the comparison operator "greater than", or ">"), and therefore more than his manager. It should be noted that the order of the rows is immaterial.

The above examples briefly demonstrate the concepts of Query-by-Example. For more details and for reformulation of the queries in predicate calculus, see Reference 1.

Q5 in linear version:

Q5. EMP (NAME:P.JONES,SAL:>10K,MGR:HARRIS)
EMP(NAME:HARRIS,SAL:10K)

OPERATIONS ON HIERARCHICAL STRUCTURES

In this section we assume that the user is familiar with the hierarchical data base model and views the data as a collection of logical hierarchical structures. He formulates his queries by filling in skeletons of the structure of the logical data base through a keyboard display with an example of a possible answer. These skeletons are initially displayed on the screen by entering their name, in much the same manner that skeleton tables are displayed in the case of a relational view.

The Query-by-Example operations are again introduced through illustrative examples, and a linear version of these examples is given for cases where a display facility is unavailable.

Let us consider a sample hierarchical data base taken from "An Introduction to Data Base Systems" by C. Date.[15] Figure 3 illustrates the various data base record types of an educational system whose function is to run a number of training courses. Each course is offered at a number of different locations.

—For each course: course number (unique), course title, course description, details of prerequisite courses (if any), and details of all offerings (past and planned)

—For each prerequisite course for a given course: course number and title

—For each offering of a given course: date, location, format (for example, duration, full-time or half-time), details of all teachers, and details of all students

—For each teacher of a given offering: employee number and name

—For each student on a given offering: employee number, name, and grade

As shown in Figure 3, there are five types of segment: COURSE, PREREQ, OFFERING, TEACHER, and STUDENT, each one consisting of the field types indicated. COURSE is the root segment type; the others are dependent segment types. Each dependent

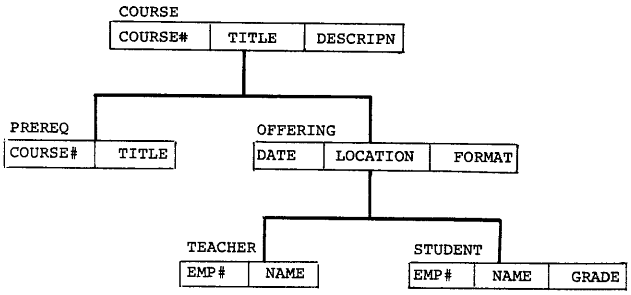


Figure 3—The education data base record type

* The 'SAVE.' & 'REMOVE.' commands were not included in the original version of Query-by-Example. They will be added to the revised version which will be published soon.[14]

has a parent—for example, the parent of TEACHER (and STUDENT) is OFFERING. Similarly, each parent has at least one child—COURSE, for example, has two.

Figure 4 illustrates specific instances (occurrences) of the data base. For example, COURSE M23 has three OFFERING occurrences, in Oslo, Dublin and Madrid. Madrid's OFFERING, in turn, has three STUDENT and one TEACHER occurrences.

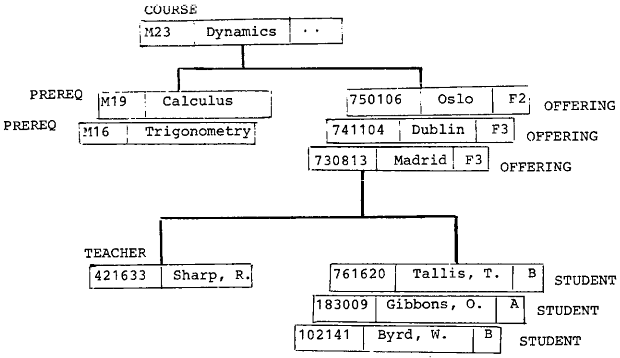


Figure 4—Sample occurrence for the education data base

We now proceed with examples of queries on this data base, some of which are also taken from Reference 15.

Q6. Get the dates of all the OFFERING occurrences where the location is Stockholm.

The user fills in the skeleton of the logical data base as follows:

COURSE

COURSE #	TITLE	DESCRIPTION

OFFERING

DATE	LOCATION	FORMAT
P.	STOCKHOLM	

Explanation:

Since there are no conditions on the COURSE fields, they are either left blank or filled in with example elements (underlined); the same is true of the DATE and the FORMAT fields in the OFFERING segment. STOCKHOLM is, therefore, the only specified constant element. The function 'P.' under the DATE field indicates that we want the dates to be printed out.

Linear version:

Q6. COURSE
COURSE;OFFERING(DATE:P. ,LOCATION: STOCKHOLM)

Explanation:

The segment name COURSE in the first row specifies the root of the logical data base. The segment name COURSE in the second row followed by a semicolon indicates that COURSE is a parent (not necessarily the immediate parent) of the OFFERING segment. The condition Stockholm Location and the function 'P.' are specified in the parentheses. It should be pointed out that since the system has the knowledge of the logical data base, in this case it is sufficient to simply put the query in the following short form.

OFFERING(DATE:P. ,LOCATION:STOCKHOLM)

The system will be able to trace the OFFERING segment to its parent segment. We originally specified the COURSE segment for clarity, so that the user will get the feel for the hierarchical tree.

Q7. List the names of all the students who achieved grade A where the offering location is Stockholm.

COURSE

COURSE #	TITLE	DESCRIPTION

OFFERING

DATE	LOCATION	FORMAT
	STOCKHOLM	

STUDENT

EMP #	NAME	GRADE
	P.ED	A

Linear version:

Q7. COURSE

COURSE ;OFFERING(LOCATION: STOCKHOLM)
OFFERING;STUDENT(NAME:P.ED,GRADE: A)

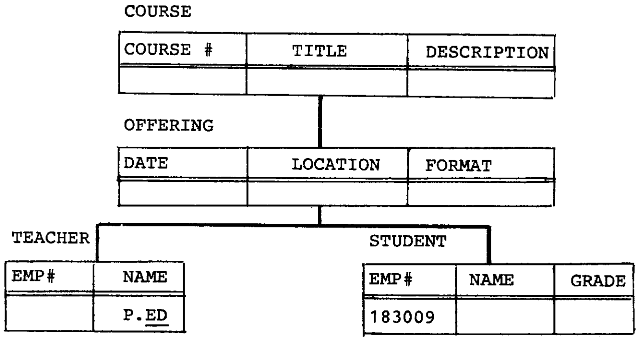
It should be noted that the sequence of the rows is immaterial. Furthermore, as the above examples illustrate, in this language, whether in the graphical or in the linear form, the user simply has to state *what* he requires to be retrieved; he does not have to be concerned with particular hierarchical sequences or hierarchical paths of the data base. In a procedural language, on the other hand, the user has to be concerned with hierarchical sequences of the data base, so that he can 'navigate' through the tree paths. As an example, let us reformulate this query in a simplified version of the DL/1 Language, which is the data sublanguage for IMS.

Q7 in DL/1 is classified as a *sequential retrieval with a conditional segment search argument* (SSA).

```
        GU  COURSE
            OFFERING
              (LOCATION='STOCKHOLM')
            STUDENT     (GRADE='A')
    NSA GN  STUDENT     (GRADE='A')
        go to NSA
```

Here the user has to be familiar with the IMS search paths in order to structure the query accordingly. In the cases where the search is forward, the formulation of the query is relatively easy; however, it becomes much more complex when a backward search path is required. One can say that here "the user is forced to devote time and effort to solving problems which are introduced by the model and are not intrinsic to the questions being asked."[15]

Q8. Get the teacher's name of an offering (of any course) attended by student 183009.



COURSE

COURSE #	TITLE	DESCRIPTION

OFFERING

DATE	LOCATION	FORMAT

TEACHER

EMP#	NAME
	P.ED

STUDENT

EMP#	NAME	GRADE
183009		

Explanation:

Here the STUDENT occurrence (EMP# 183009) and the required TEACHER occurrence *must* be linked to the same OFFERING occurrence. This link is indicated by a line connecting the three segments and can be thought of as having two relations, OFFERING-STUDENT relation and OFFERING-TEACHER relation, linked by example elements as follows.

OFFERING			TEACHER	
DATE	LOCATION	FORMAT	EMP#	NAME
x	y	z		P.ED

OFFERING			STUDENT		
DATE	LOCATION	FORMAT	EMP#	NAME	GRADE
x	y	z	183009		

Note that the *x y z* links indicate that the TEACHER and the STUDENT have the same parent OFFERING.

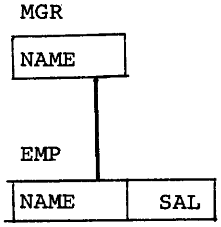
Q8 in DL/1 is classified as *use of command code F* query.

```
       GU   COURSE
    NO GN   OFFERING
       GNP  STUDENT(EMP#='183009')
       if not found go to NO
       GNP  TEACHER*F
```

Explanation:

In IMS, certain command codes are introduced to give the language more flexibility for such cases as when the user wants to search the path *backwards*. Q8 is an example of such a case. If we used GET NEXT WITHIN PARENT (GNP) TEACHER (without the code "*F"), the answer would be "segment not found", since the search would be in a *forward* direction and teachers precede students in the hierarchical sequence. What is required is a means of stepping *backwards* under the current parent and this is accomplished by the code "*F".

When we wish to relate segment occurrences to two or more occurrences of the same parent we have to qualify these parents accordingly. This is illustrated in the next example. Consider the following data base.



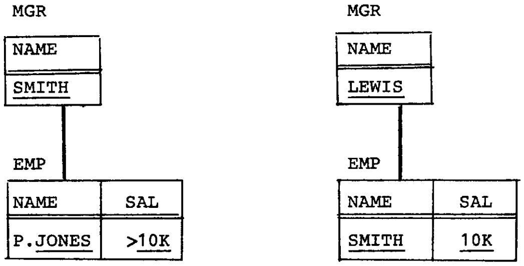
MGR

NAME

EMP

NAME	SAL

Q9. List the employees' names who earn more than their managers (same as Q5.).



MGR

NAME
SMITH

EMP

NAME	SAL
P.JONES	>10K

MGR

NAME
LEWIS

EMP

NAME	SAL
SMITH	10K

Explanation:

Here a separate skeleton is used for each occurrence of the MGR. The line connecting SMITH and JONES indicates that SMITH is JONES' manager. In order to find SMITH's salary we have to search for SMITH in an occurrence of the EMP segment; however, SMITH in the EMP segment has a manager of his own (say LEWIS) as indicated by the connecting line.

Linear version:

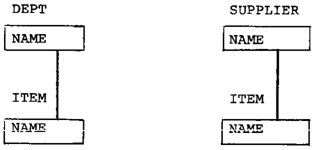
```
Q9.                  MGR(NAME:SMITH)
   MGR(NAME:SMITH);EMP(NAME:P.JONES,
                       SAL:>10K)
                     MGR(NAME:LEWIS)
   MGR(NAME:LEWIS);EMP(NAME:SMITH,
                       SAL:10K)
```

Explanation:

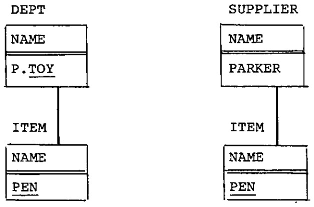
The manager qualified by the name SMITH is the parent of the employee whose name we wish to be printed out, and the manager qualified by the name LEWIS is the parent of the employee SMITH.

Note that here we cannot dispose of parent qualifications, without which the system will not be able to relate an employee to his manager.

The next query is an example of relating two different Logical data bases. Consider the following two data bases.



Q10. List the department names that sell items supplied by Parker (same as Q2.).

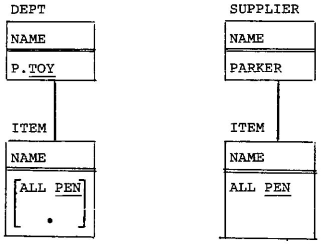


Linear version:

```
Q10.         DEPT(NAME:P.TOY)
       DEPT;ITEM(NAME:PEN)
             SUPPLIER(NAME:PARKER)
   SUPPLIER;ITEM(NAME:PEN)
```

We will end this section by a query that requires universal quantification.

Q11. List the department(s), each of which, sells *all* the items supplied by Parker.



For more details see Reference 1.

Linear version:

```
Q11.          SUPPLIER(NAME:PARKER)
    SUPPLIER;ITEM(NAME:ALL PEN)
              DEPT(NAME:P.TOY)
        DEPT;ITEM(NAME:[ALL PEN,.])
```

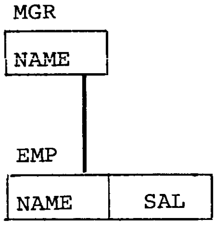
MAPPINGS

In this section we demonstrate how the user can map a hierarchical (H) view to a relational (R) view and vice versa. An example of a query involving both views simultaneously is given.

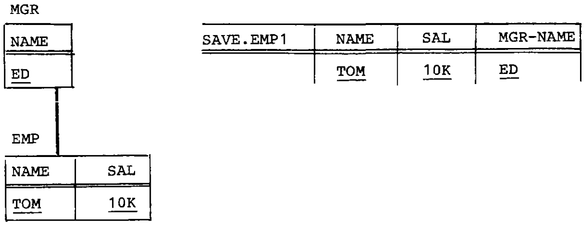
a. Mapping a hierarchical to a relational view

(H→R)

Consider the following data base.



Q12. Convert the above data base to a relation called EMP1 and save it.



Linear version:

```
Q12.      MGR(NAME:ED)
    MGR;EMP(NAME:TOM,SAL:10K)
          SAVE.EMP1(NAME:TOM,SAL:10K,
                    MGR-NAME:ED)
```

If we just want a printout in the form of a relation, without being concerned with saving it, we fill in the following skeleton.

NAME	SAL	MGR-NAME
P.TOM	P.10K	P.ED

Linear version:

(NAME:P.TOM,SAL:10K,MGR-NAME:P.ED)

This is consistent with the examples in Q3 and Q4.

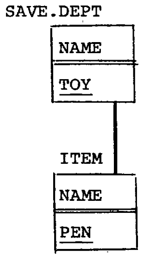
b. Mapping a relational view into a hierarchy

(R→H)

Consider the SALES relation in Figure 2.

Q13. Construct and save a hierarchical view from the SALES relation, where the root segment is DEPT.

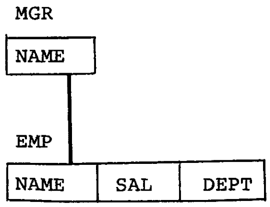
SALES	DEPT	ITEM
	TOY	PEN



Linear version:

Q13. SALES(DEPT:TOY,ITEM:PEN)
SAVE.DEPT(NAME:TOY)
DEPT;ITEM(NAME:PEN)

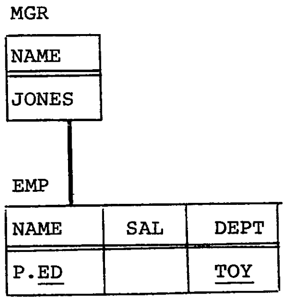
Consider the following data base which combines hierarchical and relational views.



SALES	DEPT	ITEM

The following query is formulated using both views simultaneously.

Q14. Find the employees who are managed by Jones and work in a department that sells pens.



SALES	DEPT	ITEM
	TOY	PEN

Linear version:

Q14. MGR(NAME:JONES)
MGR;EMP(NAME:P.ED,DEPT:TOY)
SALES(DEPT:TOY,ITEM:PEN)

PERFORMANCE CONSIDERATIONS

Where the performance of the system is a major factor, and it is particularly so when the data base is large, one can improve the performance by making the user cognizant of the fact that certain paths are more efficient than others; this will probably encourage him to use a more efficient approach. For example, take the TYPE relation that has ITEM and COLOR as column names and consider the following two queries.

Q15. Find the red items.

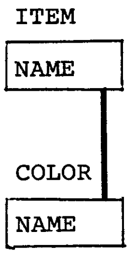
TYPE	ITEM	COLOR
	P.	RED

Q16. Find the colors of ink.

TYPE	ITEM	COLOR
	INK	P.

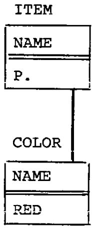
The above two queries are completely symmetric (which is, of course, one of the merits of the relational approach).

We now view the above data base as hierarchical with the ITEM segment as the root.

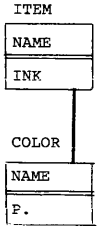


Q15 and Q16 will look as follows.

Q15.



Q16.



If the user is told that queries involving an "upward" search path, such as Q15, are less efficient than queries requiring a "downward" search path, such as Q16, he will then try to keep the number of queries formulated in the form of Q15 to a minimum. If many queries inevitably involve the same upward path, he may improve the performance by constructing another logical data base which is the inverse of the first, thus diverting the search to a downward path.

This is just a simple example of how the user can participate in the performance. It remains to be seen whether this concept can be generalized to a practical size data base, involving queries of a practical complexity.

CONCLUSIONS AND REMARKS

In this paper we demonstrated that Query-by-Example operations are independent of the view of the data base.

The feature of printing-out part of the data (such as, 'only one element' or 'the first five') is yet to be embedded in Query-by-Example. At present the 'P.' operator means 'print all the elements'. It may be possible to specify the number of elements to be printed out following the 'P.'. Example: 'P.(1)PEN' will print out one item, and 'P.(5)PEN' will print the first five items, etc.

The feature of ordering the outputted set is also not as yet included in Query-by-Example.

ACKNOWLEDGMENTS

To Peter deJong I am most grateful for his helpful discussions, suggestions, and interest throughout the preparation of this paper.

I also wish to thank my wife Rosy for correcting the manuscript and for many helpful suggestions.

REFERENCES

1. Zloof, M. M., "Query by Example," *Proc. National Computer Conference*, AFIPS press, Vol. 44, 1975, pp. 431-438.
2. Zloof, M. M., *Query by Example*, IBM Research Report RC 4917, July 1974.
3. Zloof, M. M., *Query by Example, The Invocation and Definition of Tables and Forms*, IBM Research Report RC 5115, February 1975, also appears in the proceeding of The International Conference on Very Large Data Bases, Boston, Mass., Sept. 22-24, 1975.
4. Zloof, M. M., *Query by Example: Operation on the Transitive Closure*, IBM Research Report RC 5526, July 75.
5. Thomas, J. C. and J. D. Gould, "A Psychological Study of Query by Example," *Proc. National Computer Conference*, AFIPS Press, Vol. 44, 1975, pp. 439-445. (IBM Research Report RC 5124), November 1974.
6. Scholz, K. W., *An Algorithm for the Implementation of Query-by-Example on a Small Machine*, IBM Research Report RC 5394, June 1975.
7. Zloof, M. M. and S. P. deJong, *The System for Business Automation: Programming Language*, IBM Research Report RC 5302, March 1975. Also to appear in CACM.
8. deJong, S. P. and M. M. Zloof, *Application Design Within the System for Business Automation (SBA)*, IBM Research Report RC 5366 April 1975.
9. deJong, S. P. and M. M. Zloof, *System for Business Automation: Representation of Business*, IBM Research Report in preparation.
10. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM*, Vol. 13, No. 6, June 1970, pp. 377-387.
11. Codd, E. F., "Further Normalization of the Data Base Relational Model," *Courant Computer Science Symposia*, Vol. 6, Data Base Systems, Prentice-Hall, New York, May 1971.
12. Codd, E. F., "Normalized Data Base Structure: A Brief Tutorial," *Proc. 1971 ACM SIGFIDET Workshop on Data Description, Access and Control*, San Diego, November 1971.
13. Niebuhr, K. E. and S. E. Smith, *Implementation of Query-by-Example on VM/370*, IBM Research Report in preparation.
14. Zloof, M. M., *Query-by-Example (revised) including a linear version, Formal Syntax and Semantics*, IBM Research Report in preparation.
15. Date, C. J., *An Introduction to Data Base Systems*, Addison-Wesley, 1975.

A virtual memory system for a relational associative processor

by S. A. SCHUSTER, E. A. OZKARAHAN and K. C. SMITH
University of Toronto
Toronto, Ontario, Canada

ABSTRACT

The Relational Associative Processor (RAP) is an experimental "backend" cellular processor for implementing data base management systems. RAP is particularly well suited to supporting Codd's relational model of data. The capacity of a RAP device implemented with current IC and memory technology can be estimated to contain 10^8 to 10^9 bits of associatively processable data. Because many data bases are larger, a virtual memory environment for RAP has been proposed and its performance simulated. The environment incorporates conventional memories for bulk storage and a single RAP processor—both controlled by a general purpose front-end computer. The system requires that the entire relational data base be divided into pages of size equal to one RAP cell memory. A buffer memory is added to RAP to permit the overlap of paging with processing. It has been found that user environments containing small relations or queries exhibiting either long processing times relative to paging requirements or some "locality" (defined as the degree to which sequences of queries reference some relations more than others) can efficiently page data between large data bases and data base machines without significant losses in performance.

INTRODUCTION

The relational model of data is an important approach to data base management because it presents its user with a simple, consistent, and operationally complete view of data.[1] However, its high-level user-oriented view creates serious problems for efficient generalized implementation on conventional hardware. We briefly review the relational model and a special purpose peripheral processor called RAP which is designed to provide high performance relational data base operations.[2–4] A small prototype version of RAP is being implemented at the University of Toronto. This paper concentrates on the problem of using RAP to support large relational data base applications. To do this, we propose several architectural extensions and a software support system to create a virtual memory environment for a RAP processor. A model of the virtual memory environment has been simulated and the results are presented and analyzed.[5]

THE RELATIONAL ASSOCIATIVE PROCESSOR "RAP"

A relation can be viewed as a table whose rows contain data about a set of similar entities. The table's or relation's name identifies the set of entities being described and the column headings are the names of the attributes which are used to describe the characteristics of each entity. Each row contains an n-tuple of values—one for each attribute—which uniquely describe each entity. The set of values that can be taken by an attribute is called a domain. A relational data base is composed of a collection of time-varying relations that may change because of modifications, insertions, and deletions. Several relations can be interrelated through common domains to formulate complex queries.

The Relational Associative Processor (RAP) is an experimental "backend" non-numeric processor designed to efficiently implement data base management systems. Its architecture and instruction set is particularly well suited for supporting the relational model of data. In addition, recent research indicates that RAP is sufficiently generalized to support set-oriented hierarchical and network views of data. This paper concentrates on RAP's relational aspects.

RAP architecture is based on the observation that relational operations of search, retrieval, statistical computation, and update are inherently associative and set-oriented. The basic organization of RAP is shown in Figure 1. The design incorporates an array (i.e., parallel set) of circularly connected associative cellular processors which are driven by a central controller and statistical processor. A RAP instruction is a data base operation which is executed by each cell in parallel directly on the cell's memory. Each cell is composed of a microprocessor and a sequential circulating memory (e.g., a track of a drum or disk, CCD's, bubble memories, etc.). The rows of a relation are stored as blocks of data on one or more cell memories. The RAP data structure allows rows to be duplicated.

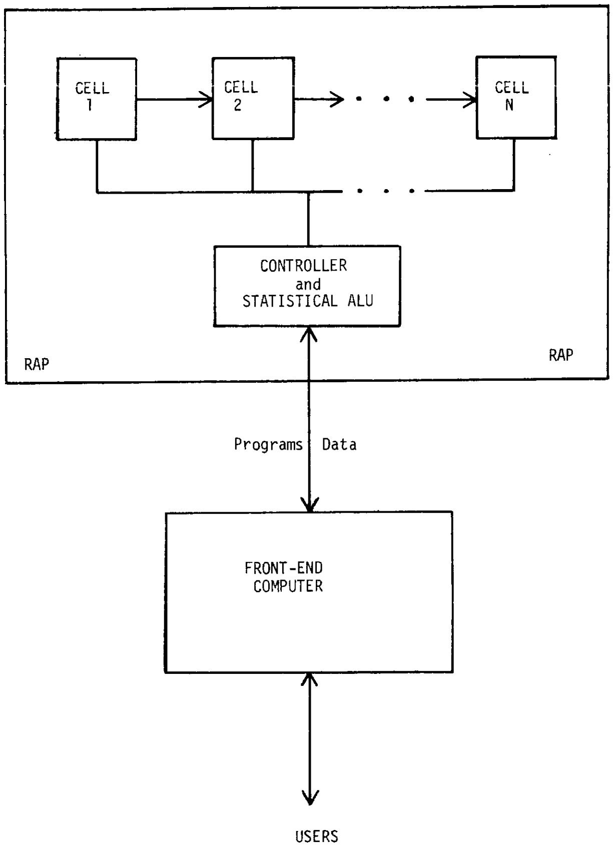


Figure 1—Basic organization of RAP architecture

Relational queries can often be constructed from just a few RAP primitives and often only a single RAP primitive is required. The RAP system is designed to execute its most important instructions within one parallel rotation of the cell memories. Because operations are accomplished in parallel on the entire storage, RAP is an effective associative processor providing significant search and manipulation efficiencies. That is, RAP accomplishes relational data base management without complex data structures and software aids such as inverted lists and hashing for multi-key searching required in conventional systems. This is especially important for applications which have extensive update activity. The extra indices and ordering requirements must be maintained whenever an update occurs in a conventional data base system. In RAP, only the relation itself has to change and this is accomplished directly on the data without having to move any portion of the data base to the front-end processor.

An analytical model was constructed which compared the performances of RAP to a conventional data base management system.[4,6] The model considered resident data bases for RAP and fast access paths in the form of inverted lists for the conventional system. The results showed that significant gains in query execution speed can be achieved by the RAP architecture over the conventional system even if it uses fixed-head disk memories. The model includes queries of the form: simple Boolean selection retrievals or updates, retrievals including statistical criteria in the query qualification, and retrievals involving implicit joins. This study indicates that, under many circumstances, on-line retrieval and updates of large data bases may only be possible with the aid of RAP-like systems.

The concept of providing large scale associative processors and memories for data base management are also currently being explored by others.[7-10] These devices are generally referred to as "data base machines". Most of the discussion to follow may also be applicable to the general theory of data base machines.

The maximum capacity of a single data base machine implemented with current IC and memory technology can be estimated to contain from 10^8 to 10^9 bits of associatively processable data.[4] Data compression, of course, should be exploited. This capacity may be sufficient for many applications but there are others which require larger storage. The costs of data base machines may not permit them to be casually duplicated when larger storage is required. This then raises the question as to what architectural extensions and software techniques can be explored to efficiently extend the address space of RAP-like data base machines. A virtual memory system for RAP is proposed which incorporates conventional disk memories for bulk storage and a single RAP processor—both controlled by a front-end general purpose computer.

RAP ARCHITECTURE FOR LARGE RELATIONAL DATA BASES

The need for the virtual memory system arises when the data base is larger than RAP memory capacity. However, it is assumed that all the pages for a query can be contained within RAP memory. This assumption is reasonable in light of the expected memory capacity for RAP (10^8 to 10^9 bits). For applications requiring very large relations, partitioning of large relations into smaller ones containing the same columns may be required. This would require a user to create a sequence of tasks to process several subqueries over the smaller relations and assemble or interrelate their results.

The proposed virtual memory system requires that the entire data base be divided into fixed size pages equal to the capacity of one cell. A page contains rows from only one relation. Each page has a unique identification. Therefore, the set of rows for a complete relation can be specified by indicating the pages that contain the rows.

The extensions to RAP architecture for implementing a virtual memory system are displayed in Figure 2. In this configuration, the front-end general purpose computer contains a virtual memory monitor and acts as an I/O interface for paging data between conventional bulk memories (e.g., direct access secondary memory devices) and RAP. A large portion of the data base resides on the bulk memories and only the "working set," that is the collection of pages containing the relations required by the currently processing queries, resides in RAP memory.

In order to reduce paging overhead, overlapping of paging I/O with query processing is essential. To achieve this, each cell has been extended to contain two memory components. At a given time, one serves as the active or processor memory while the other acts as the buffer or alternate memory. The memory serving as the active memory is used in query processing while the other acts as the I/O buffer to permit paging to take place concurrently with query processing. The roles of the memories can be reversed for each cell independently as required during the operation of the system. Thus, RAP logically contains two cyclic memories—one serving as the active associative memory while the other is a buffer. The hardware logic for switching between cell memory pairs has been designed in detail for RAP.[4] The extra logic required for doing this adds little to the complexity of the overall system.

The RAP controller is still connected to the front-end computer via a dedicated channel to receive programs and transfer results of users' requests. However, the current buffer is driven by a separate I/O controller which is connected to the front-end computer by a separate channel. The paging to and from the buffer is accomplished by standard channel programs before a new query is processed.

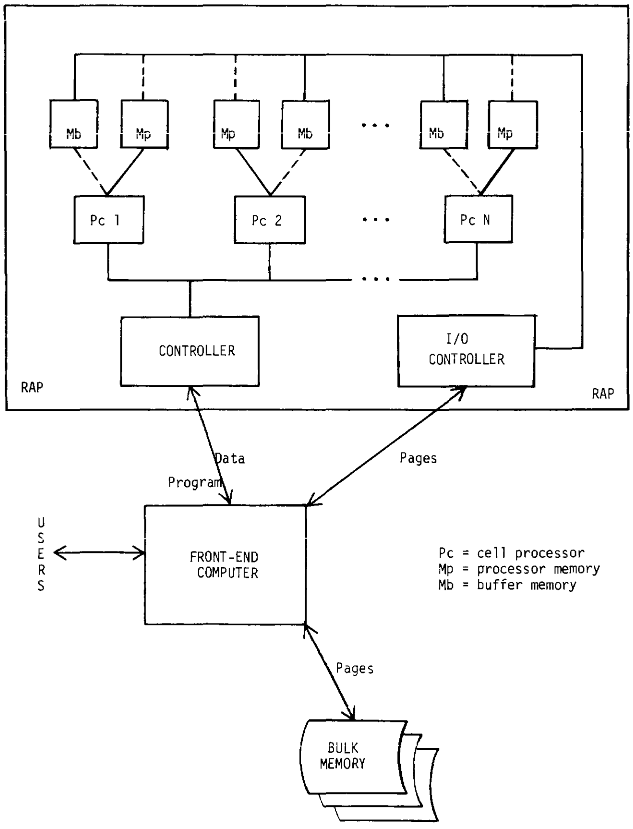


Figure 2—Hardware architecture of the virtual memory system

The paging for a waiting query is overlapped with the processing of a currently executing query. Lookahead paging is possible because RAP associatively processes all the pages of the relations referenced in a query. Since each query must specify which relations are to be processed, it specifies which pages are to be processed at the same time. Once the required pages are transferred and execution of the current query is finished, the roles of the memory pairs can be switched.

There are two other features of the RAP design which are not shown in Figure 2 that help to increase the efficiency of the virtual memory system simulated in this study. They are the fast read-out scheme and the multiprogramming facility.[4] The fact read-out scheme for retrieval queries interleaves selected tuples from different cells during a memory revolution to maximize the channel utilization. The multiprogramming facility allows queries to be divided into two priority classes and executes them with respect to a preemptive scheduling discipline to provide a foreground-background query execution facility. This allows the RAP processor to control query execution by not allowing a long query to tie-up the device. The logic for implementing hard-wired two-task multiprogramming requires the duplication of less than 10 percent of the cell hardware.[4]

A VIRTUAL MEMORY SYSTEM

Locality

The virtual memory system to be discussed is designed to exploit user characteristics that affect total system performance with respect to paging. In conventional virtual memory systems, the important property of locality of address references in a user program is used to reduce paging. In an analogous way, we define "locality of query references" or simply "locality" as a phenomenon which is manifested by a nonrandom distribution of references to the relations in a data base during the execution of a sequence of queries. High locality is reflected when a small subset of the relations in a data base are referenced more often than the other relations, i.e., the frequency of relation reference is described by a distribution with low variance. The effect of varying degrees of locality is studied in the simulation.

Unfortunately we lack experimental data and are not aware of any studies performed in connection with locality in relational data bases. However, we will present some ad hoc arguments for its existence. Locality will be loosely examined with respect to four classes or environments of processing.

There are many data processing tasks that occur in a reservation system environment such as for hotels, airlines, shipping, etc. As a deadline approaches for an entity, e.g., the scheduled time for a plane flight, the activity in the data base is expected to become concentrated within the area representing this entity. If there exists a relation for each flight which records the passenger reservations for a particular flight, then the access to those relations scheduled to depart shortly will have greater frequency of reference resulting in high locality.

A second argument for high locality is that the use of high level interactive query languages encourages "browsing". In such an environment, a user can interactively search for relevant data by step-wise refinement of queries on the same relations. The iterative refinement of queries can cause the same relations to be accessed repeatedly during a short period of time.

In a data processing environment using batch multiprogramming, we expect that queries are generated within the constructs of host language programs. These programs are usually involved in the generation of complex reports or batch updating. The logic of such programs is often repetitive which cause queries to be embedded within program loops. This could cause sequences of queries to concentrate references over a subset of the relations in the data base.

A fourth argument for high locality is that certain relations may play an important role by interrelating other relations. These relations have a high percentage of attributes whose domains are common to other relations. These relations must be referenced extensively to link data in different relations in the formulation of many queries. Because these "linking relations" may be referenced often, queries over such a data base could exhibit high locality.

The system presented was designed to exploit high locality characteristics when they exist. The results will show that a modest amount of locality can greatly extend the processing efficiency of a data base machine for processing data bases which are large relative to the capacity of the machine's memory.

The virtual memory system structure

The principal modules and data flow of the virtual memory system are shown in Figure 3. The overall control of the system is concentrated in a central MONITOR. The MONITOR is also responsible for communicating the block of individual operations of a query to drive the RAP processor.

As jobs enter the system they are classified by the CLASSIFIER module as either class-1 or class-2. Class-1 jobs are queries that have high priority and are to be processed uninterrupted such as on-line retrievals or any update. Class-2 jobs are retrieval queries with long processing times which can be run in the background, interrupted by class-1 jobs, and then resumed when no class-1 jobs are waiting to be processed. We will assume that all class-2 jobs can be resumed from their point of interruption regardless of possible effects of an intervening class-1 update job. If the semantics of the query is such that it must be completed in its entirety before any of its relations can be updated, it would be classified originally as a class-1 job. The data for each job that determines its class can be either found by examining the query's instructions or by user specified parameters. We further assume that all jobs classified as class-1 can be reordered during a small segment of time. (This policy has been made over-simplistic for purposes of the simulation. It is acknowledged that updates must often execute in a specific order.)

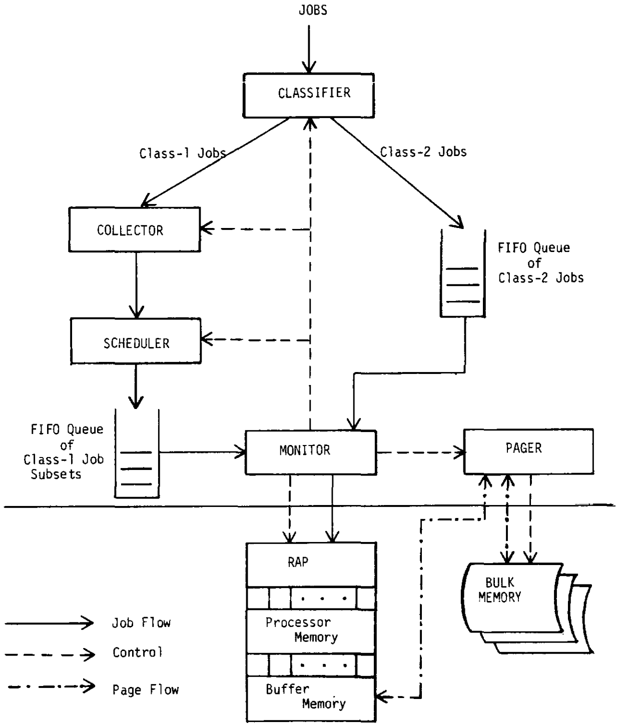


Figure 3—Structure of the virtual memory system

After classification, class-2 jobs simply enter a class-2 FIFO queue. Class-1 jobs, however, are kept in a job pool by the COLLECTOR module. Under the direction of the MONITOR, all class-1 jobs in the pool will be released to a module called the SCHEDULER. Two policies have been suggested for controlling the COLLECTOR. Only the first has been simulated. In this case, class-1 jobs are collected as long as there exist previously scheduled class-1 jobs that are waiting and/or processing. When such jobs no longer exist the COLLECTOR releases its jobs to be scheduled. If no class-1 jobs have been collected, then the current class-2 job is started or resumed if a preempted one exists. The class-2 job is allowed to process until the next class-1 job is received and preprocessed. The single class-1 job is immediately scheduled and, when its paging requirements have been satisfied, it then preempts the class-2 job and acquires the processor.

The second policy is to let the COLLECTOR collect jobs for a fixed amount of time which would be determined by the MONITOR. In retrospect, it is felt that this would have given better results because larger pools will be presented to the SCHEDULER which is designed to exploit high locality by reducing page faults through job reordering.

The SCHEDULER receives the pool of class-1 jobs and orders them by first finding the job which causes the least amount of page faults with respect to the current contents of RAP memory. This can be done because the relations referenced in the queries are known by the MONITOR. It continues to order the rest of the jobs by finding the next job whose pages most overlap RAP and so on. The ordered pool of jobs are then partitioned into an ordered set of job subsets such that all the relations for each subset can be contained within the data base machine memory. The ordered subsets are then placed in a FIFO queue to wait for the processor.

The final module to be discussed for the virtual memory system is the PAGER. It is responsible for the paging or buffering strategy and the page replacement policy. Paging is performed on a non-demand basis. This is possible because the pages required by a subset are known before processing starts. The demand paging philosophy encountered in programming systems does not apply here, because, having job subsets sitting in the queue, one can start the processing of a current subset and at the same time start paging data into the buffer for the following subset. This scheme is aimed at reducing the average amount of idle processor time that a subset causes due to paging.

The replacement policy for pages in the data base machine memory is to first replace any pages that have not been updated and are not referenced by jobs in the ordered subsets waiting in the queue. If more space is required, then those referenced farthest in the job subset are replaced. An actual page-out occurs only if the replaced page was updated, otherwise it is simply overwritten in the buffer. Page-outs occur after the role of the cells' memories, which contain the pages to be replaced or paged-out, are switched from active memory to buffer memory.

The effectiveness of the SCHEDULER and PAGER in exploiting locality is measured by an index called "system locality (SLOC)". SLOC is defined as follows. Let pg[i] be the number of unique pages that are required by subset i and pgcom[i] be the number of pages in common between subset i and the contents of RAP memory at the time when paging for subset i is to take place. Thus,

$$\text{SLOC} = \sum_{i=1}^{n} \text{pgcom}[i] / \sum_{i=1}^{n} \text{pg}[i]$$

for i = 1,2, . . . , n consecutive class-1 job subsets. The system locality will be near one for environments requiring little paging between subsets and near zero for environments requiring excessive paging. System locality depends on the degree of locality, the number of pages in the entire data base relative to the number of pages that can be contained in RAP memory, and the size of the relations in the data base.

SIMULATION OF AN ON-LINE ENVIRONMENT

Lacking experimental user data, an on-line environment has been hypothesized and simulated for the described virtual memory system.[5] The data base machine is modelled with the expected parameters for a moderately large size RAP processor. The simulation was coded in GPSS on an IBM 370/165.

The on-line environment

The application environment studied could be conceptualized either as an on-line reservation system for hotels or airlines or a parts distribution and inventory system. Most of the jobs fit the class-1 description with a few class-2 background batch jobs representing management reporting, billing, financial accounting, payroll, etc. The arrival rate of queries is fast and most require short processing times. Most queries involve only a few relations. The following is a list of parameters used in the experiments. The times are given in terms of the number of revolutions or circulations of the RAP memory.

RAP:

(a) 1 revolution (rev) = 50 milliseconds
(b) RAP processor = 200 cells
(c) RAP memory capacity = 400 pages
(d) active memory = buffer memory = 200 pages
(e) page size = $0.5*10^6$ bits
(f) paging rate = 1 page/rev

Data base:

(a) total data base size = 2000 or 5000 pages
(b) number of relations for short jobs = exponential (mean = 1.5 relations)
(c) number of relations for long jobs = exponential (mean = 3 relations)
(d) number of pages per relation = 5, 10, 20, or 25 pages

Locality:

(a) low—modelled by a uniform distribution
(b) medium—modelled by an exponential distribution
(c) high—modelled by a hyperexponential distribution

As each job (query) is generated in the simulation, one distribution is referenced, depending on the locality being simulated, to determine the relations needed by the job.

Arrival and processing time distributions:

(a) job interarrival time=Poisson (mean=40 rev/job)
(b) processing time for short jobs=exponential (mean=4, 8, 16, or 32 rev)
(c) processing time for long jobs=Erlang-2 for class-2 and exponential for class-1 (both with mean=104 rev)
(d) processing time for class-1 jobs=hyperexponential (due to (b) and (c))
(e) processing time for class-2 jobs=processing time for long jobs (due to (c))

Job proportions:

(a) proportion of class-1 vs. all jobs=0.90
(b) proportion of updates vs. all jobs=0.40
(c) proportion of long updates vs. all jobs=0.03

The above parameters were chosen in such a way as to create a system load which guarantees that a class-1 and/or class-2 job would almost always be present in the job mix. Thus, the RAP processor in this experiment would rarely be idle due to empty job queues. However, idle processor time would be caused by extensive paging requirements.

Simulation results

The goal of this study was to determine the relative effects on the average time-in-system, referred to as the response time, for a class-1 job when the following were varied:

(a) short job mean processing time
(b) number of pages per relation
(c) locality
(d) data base size

Also studied was the effect of these variables on system locality which reflects the paging performance from the system's point of view.

Figure 4 shows the effects of processing time for short jobs (proctime), locality, and data base size (dbsize) on RAP system response for class-1 jobs. For this experiment the number of pages for a relation was fixed at 10. The response time, RTIME, indicates the average time a class-1 job spent in the system. The results can be approximately summarized as a family of straight lines of the form "y=a*x+b". For the experiment we get:

RTIME:=proctime+((log dbsize)/locality)

where ":=" means "proportional to." That is to say, response time is directly proportional to processing time, as would be expected, with an upward shift depending on locality and data base size. The line "y=x" is shown because it represents the theoretical optimum for the environment simulated. The average response time can never be less than the average time it takes to process a job.

The somewhat linear upward shift in the curves away from the optimum is due to idle processor time because of paging. The paging requirements are reduced by high locality, but increased in a diminishing way (this will be explained later) when the total data base size grows larger relative to RAP memory capacity.

It should be noted that the curves slowly converge toward the optimum for longer processing times. This is to be expected since the number of pages for a relation are fixed in this experiment. Longer processing times allow more paging to overlap with processing and, when the processing times are long enough, there will be time to do all paging before the next job subset is to be processed.

Figure 5 shows the effects of relation size in terms of the number of pages (npgrel), locality, and data base size (dbsize) on RAP system response time for class-1 jobs. For this experiment the average processing time for a short job was set at 8 revolutions. The results can be summarized as follows:

RTIME:=((log dbsize)/locality)*npgrel+proctime

That is to say, response time is directly proportional to the size of relations and the slope of the relationship is affected by locality and data base size. The line "y=proctime" is shown since it represents the average experimental optimum as discussed earlier.

The fact that response time is directly related to the

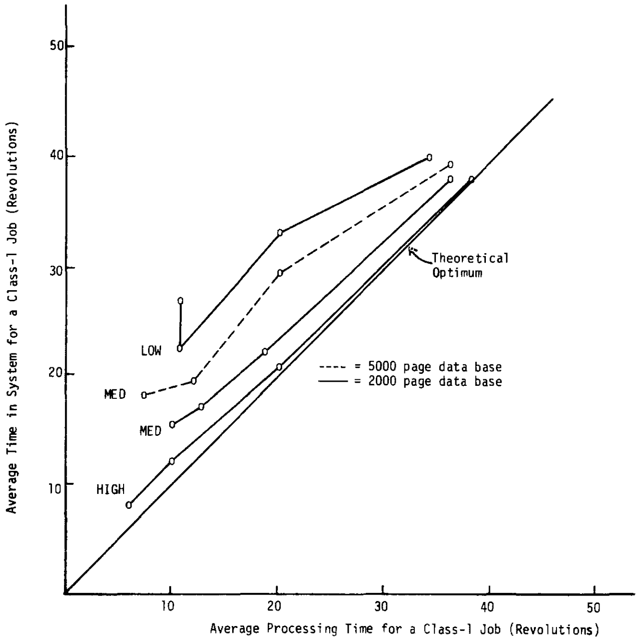


Figure 4—The effect of locality, processing time, and data base size on system response

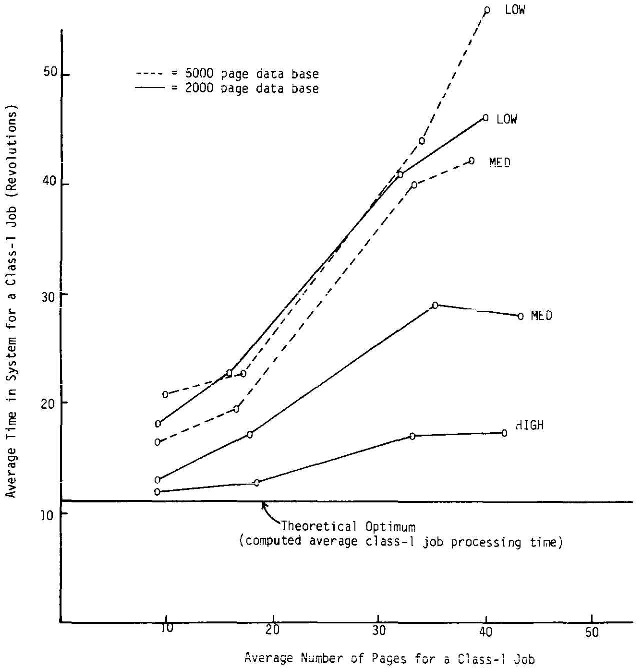


Figure 5—The effect of relation size, locality, and data base size on system response

average number of pages occurring in relations of the data base is not surprising. Since all pages of a relation are required to be in RAP memory to be processed, the more pages required, the more paging is to be expected. Given a fixed average processing time, we expected and also observed that response is degraded when large relations must be paged.

However, the effect of locality can be profound because it affects the slopes of the response time curves. When the locality is higher, there is a good chance that some of the relations currently in RAP memory will be needed by the following job subset. Therefore, the effect of paging larger relations will not be so dramatic.

Data base size also affects the slope of the response time curves. As the data base gets larger relative to RAP memory, the effect of locality is diminished. That is, the same pattern of locality will be spread over more relations causing the locality to appear more random. It should be noticed that for the low (random) locality curve, the effect of increasing the size of the data base was very small. It is observed that paging due to random locality will be independent of data base size. When extensive paging is required because of random reference, it does not matter what size of pool contains the pages.

Figure 6 shows the effects of these experiments on the average system locality (SLOC). The data from which these curves are derived also showed that system locality is independent of a job's processing time or relation size for a fixed locality distribution. The results can be summarized as follows.

SLOC := (1/(log dbsize))*locality

That is, the higher the locality the better the system will perform since less paging will be expected. However, as the data base size grows, the effect of locality will diminish and performance will approach that of a random locality environment.

SUMMARY AND CONCLUSIONS

The Relational Associative Processor (RAP) is an experimental "backend" non-numeric processor for implementing a data base management system which supports Codd's relational model of data. RAP architecture is based on the observation that relational operations of search, retrieval, statistical computation, and update are inherently associative and set-oriented. The design incorporates an array (i.e., parallel set) of interconnected associative cellular processors which are driven by a central controller and statistical processor. Each cell is composed of a microprocessor and a sequential circulating memory (e.g., a track of a drum or disk, CCD's, bubble memories, etc.). Each data base operation is executed within the cells which operate in parallel directly on their memories. RAP accomplishes a complete set of data base operations without the need for indexing and its associated software and maintenance.

The capacity of a RAP device implemented with

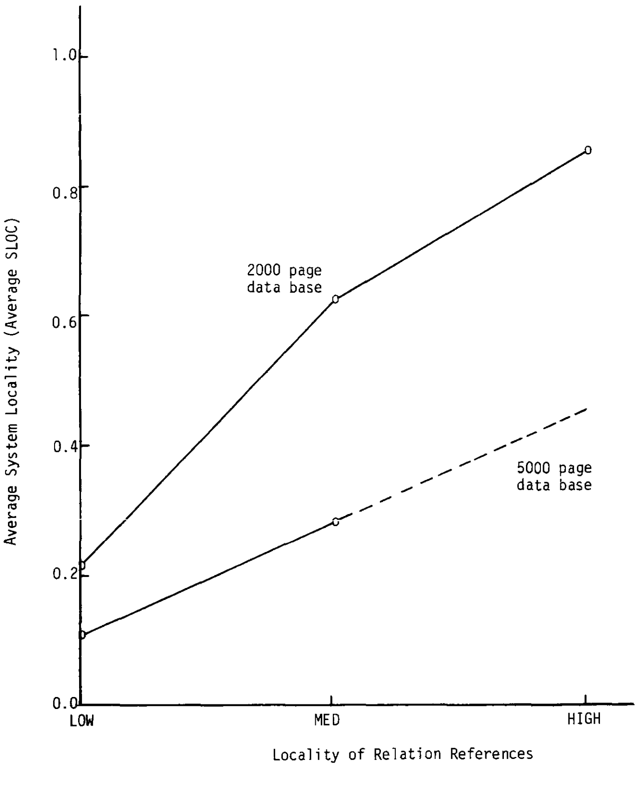


Figure 6—The effect of locality and data base size on system locality

current IC and memory technology can be estimated to contain from 10^8 to 10^9 bits of associatively processable data. Because of this limitation, a virtual memory environment for RAP has been proposed and its performance simulated. The environment incorporates conventional memories for bulk storage and a single RAP system—both controlled by a front-end computer. An entire relational data base is divided into pages of size equal to one RAP cell memory. A buffer memory is added to RAP to permit the overlap of paging with processing.

Each RAP program represents one relational query (job) on one or more relations. The system maintains an ordered queue of waiting queries. Before the next selected query is processed on RAP, a table look up is made to determine if all the pages for the query are present in RAP memory. If any pages are not currently residing in RAP, they are paged in replacing the pages for the relations referenced farthest in the job queue and/or those not needed at all such that pages that have not been modified are replaced first. The jobs waiting to be processed are scheduled by ordering them into a queue in such a way that paging is reduced. It has been found that user environments which contain small relations or process queries exhibiting either long processing times relative to paging requirements or some "locality of relation reference," can efficiently page data between large data bases and data base machines without significant losses in performance.

ACKNOWLEDGMENT

The authors are happy to acknowledge the contribution of Raymond Nakano who developed the simulation program and made many valuable comments throughout the preparation of this paper.

This work has been supported by the Federal Department of Communications, the Department of Supplies and Services, and the National Research Council of Canada.

REFERENCES

1. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *CACM 13*, 6, 1970, pp. 377-87.
2. Ozkarahan, E. A., S. A. Schuster and K. C. Smith, *A Data Base Processor*, CSRG-43, Computer Systems Research Group, University of Toronto, September 1974.
3. Ozkarahan, E. A., S. A. Schuster and K. C. Smith, "RAP—An Associative Processor For Data Base Management," *AFIPS Conf. Proc.*, Vol. 44, 1975, pp. 379-87.
4. Ozkarahan, E. A., *An Associative Processor For Relational Data Bases—RAP*, Ph.D. Thesis, Dept. of Computer Science, University of Toronto, January, 1976.
5. Nakano, R., *A Simulator for a RAP Data Management System*, M.Sc. Thesis, Department of Computer Science, University of Toronto, 1976.
6. Ozkarahan, E. A., S. A. Schuster and K. C. Sevcik, *Performance Evaluation of a Relational Associative Processor*, CSRG-65, Computer Systems Research Group, University of Toronto, January 1976.
7. DeFiore, C. F. and P. B. Berra, "A Data Management System Utilizing an Associative Memory," *AFIPS Conf. Proc.*, Vol. 42, 1973.
8. Moulder, R., "An Implementation of a Data Management System on an Associative Processor," *AFIPS Conf. Proc.*, Vol. 42, 1973.
9. Su, S. Y. W. and G. J. Lipovski, "CASSM: A Cellular System for Very Large Data Bases," *Very Large Data Bases Conf. Proc.*, Vol. 1, No. 1, September, 1975.
10. Lin, C. S., D. C. P. Smith and J. M. Smith, "The Design of a Rotating Associative Memory for Relational Data Base Applications," to appear in *TODS*, Vol. 1, No. 1, 1976.

Managing the census data base—Data description, acquisition, and manipulation

by JAY-LOUISE WELDON
New York University
New York City, New York

ABSTRACT

The U.S. Bureau of the Census collects and maintains a broad base of socio-economic and demographic data on population and housing in the United States. Summarizations and statistical information from this data base are made available to the public in printed reports. In addition, portions of the data are made available in machine-readable form, and thus are appropriate input for further research studies.

For most potential users, however, many obstacles exist in making use of the machine-readable Census data. These obstacles may be classified into three areas: the identification and selection of relevant data items from the available data to support a given research objective, the acquisition of accurate and complete data, and the physical storage and processing of such a large data base.

This paper discusses each of these problem areas and identifies the factors involved. Illustrations of possible solutions are presented, based on the author's experience with the Census data base at the University of Pennsylvania. Guidelines for dealing with such problems are provided for other prospective users. Finally, some general recommendations are made for avoiding or reducing these types of problems in future Census data products.

INTRODUCTION

The basic Census data base, as maintained by the U.S. Bureau of the Census, contains the actual (or encoded) responses for each individual who completed a Census questionnaire in the 1970 Census of Population and Housing. Portions of this data base are made available to the public in three forms:[9,10,11]

(i) *printed census reports,* which summarize various responses for certain geographic areas;

(ii) *summary data tapes,* which also contain response data summarized by geographic area but in computer-readable form; and

(iii) *sample data tapes,* called the Public Use Sample, which contain selected data records from the basic Census data base, also in computer-readable form.

The printed reports are the most frequently available and immediately usable of these three forms. These reports are available in most libraries and provide summarizations of basic housing or population characteristics for the most commonly accessed geographic regions. For example, from the reports a researcher may gather data on counts of housing units or on population by age or sex for counties or states. While convenient, the printed reports may not suffice for many researchers. For example, data may be required on more specialized data items, or by smaller geographic areas. For computer analysis, the researcher may wish to obtain a large volume of data records in computer-readable form. In these latter cases, the summary tapes and the Public Use Sample are required as data sources.

The summary data tapes, organized into six parts called "Counts," contain the summarizations from which the printed reports were developed. The six Counts differ with respect to the data tabulations they contain and the size of the geographic areas for which these data are summarized. Each tabulation represents the result of cross-tabulating one or more questions from the Census questionnaire. As such, more than one data item (or cell) may be contained within one tabulation. For example, the tabulation "Population by Race" requires four data cells, corresponding to three racial groups plus the total. The geographic area for which the tabulations are summarized are identified in each record by a set of codes. The summary data tapes contain summarizations for states, counties, and Standard Metropolitan Statistical Areas (SMSA's), as well as for smaller areas, such as census tracts, blockgroups, and Minor Civil Divisions (MCD's).

The Public Use Sample tapes contain data records of actual questionnaire responses with most identifying information removed. These records are selected according to different sample sizes and for different geographic areas, and grouped into six files according to these two characteristics. The data items in the Public Use Sample records are coded responses rather than counts or tabulations. However, coded geographic information is used to identify each record, as in the summary data.

The decision to use and/or acquire either type of computer-readable Census data forces the researcher/user to confront a number of problems. The problems encountered can be classified into three general areas: data description, acquisition, and manipulation. Problems in data description arise when the user attempts to identify and/or select the data items and geographic areas most appropriate for his/her research. Data acquisition problems center around the user's ability to obtain complete and accurate data in a form compatible with his/her computer installation. Manipulation problems are largely due to the overwhelming size of the Census data base (over two thousand reels of computer tape, as originally produced from the Bureau of the Census). The user/implementor is faced with questions of how to store and access this large data base in the least expensive fashion.

This paper addresses each of the three problem areas and presents guidelines for potential census users in each area. These guidelines are based on experience with the implementation and use of the Census data base in a university environment. In addition, recommendations are made for improving future Census data products and services.

IDENTIFICATION OF DATA TABULATIONS AND AREAS

Before evaluating alternatives for Census data acquisition and manipulation, the prospective user must be able to identify and select those data items which are required for his research. The two-dimensional logical structure of the computer-readable Census data base requires that a potential user select both the *tabulations* he wishes to use and *geographic areas* for which these tabulations are summarized.

Selection of tabulations

Problems in identifying and selecting appropriate tabulations from those available arise due to (i) duplication of variables, and (ii) questions of data reliability. The same variable (e.g., housing unit value) may appear in several summary data Counts, though in a slightly different context in each (see Table I). User aids, such as the *1970 Census Data Finder*, should be consulted to locate the tabulations that may be of interest under more general subject headings. In every case, the user should refer to the formally defined Census Concepts[9] to insure proper interpretation of the terms used, e.g., "mixed parentage" under race.

TABLE I—Representative Tabulations Relating to Value of Housing Units in the Census Summary Data

Item	Tabulation Number	Count
Aggregate $ value (of housing unit)	17	First
Units for which value is tabulated by value class, by race	35	First
Aggregate $ value for units with all plumbing facilities	27	Second
Value, type of household, age and race of head	57	Second
Aggregate $ value	1	Third
Value	22	Third
Aggregate $ value	1	Fourth—Housing
Value, occupancy status, and race of head	52	Fourth—Housing

Reliability problems include questions of sampling, allocations, and actual errors in the data. Since a minimal number of questions (e.g., five on Population) were asked of all Census respondents, all other tabulations are based on questions asked of a sample of respondents. These sample counts are weighted to approximate the total, but actual totals and weighted totals may not agree in every case. Further, users of tabulations or small geographic areas, such as block-groups or enumeration districts, may find that the totals on a sample question reflect the responses of an unacceptably small number of actual responses. For example, a 5 percent sample item on a 300-household census tract would reflect only 15 actual households. For items that are further divided, e.g., into rental or owner-occupied, the count will be even lower.

"Allocations" are substitutions made by the Bureau of the Census for data items missing from the original Census responses. This procedure is documented and data indicating the allocations that were made exist within the Census data base. However, a user who makes a hasty selection of tabulations without investigating the impact of these pseudo-data may be jeopardizing his results. A high proportion of allocations for any data item affects the reliability of all tabulations utilizing that item.

Certain tabulations may be missing entirely from the summary data tapes. In cases of small area tabulations, the Bureau of the Census often suppresses data items that may impair confidentiality. For example, income tabulations might be suppressed for block groups containing only two individuals older than 65. These situations are indicated by a series of suppression codes placed into the data record at the time of suppression. The user must simply be aware of their meanings and make sure any access software handles these zero-filled fields accordingly.

Incorrect tabulation data discovered by the Bureau of the Census are generally reported in Bureau publications, such as the *Data User News* (formerly called

Small Area Data Notes). However, it behooves the user to be alert to these announcements, since error correction is the responsibility of the owner of the data, not the Bureau. Further, such errors are reported as they are discovered and are not reflected in the primary documentation accompanying the receipt of Census data. Thus, a diligent user is forced to scan back issues of Census publications to spot errors that may affect his analyses.

Selection of geographic areas

In selecting geographic areas, the user is faced with problems involving geographic codes, record-sequencing, area matching, and missing data. All geographic areas represented in the Census summary data are identified by numerical codes. The Census Bureau provides a user aid called the Master Enumeration District List (MEDList) in which each code is associated with the corresponding state, county and area or place names. In order to access the summary data for any area, the user must know the codes of the areas desired, since no names exist on the tapes. While codes are also used in the Public Use Sample, no corresponding list exists for these codes. To further complicate the issue, the codes used in the P.U.S. are not always the same as those in the summary data.

A user who is dealing with regions or areas that do not coincide with census geographic areas must match the census designations with the region desired. Doing this may require overlaying the boundaries of these "special" areas on Census maps or using the Address Coding Guide (ACG). The latter is a list of block faces and their associated census tract designations originally used to properly sort Census responses. Using the ACG may enable a user to convert his "special" regions into tabulated areas. Or, at worst, he can describe these regions in terms of addresses and request a special tabulation from the Bureau.

Even for fairly standard geographic areas, the sequencing of the summary tapes may prove an obstacle to easy access. A "sequencing key" field, made up of various geographic codes plus category codes added for the summary, exists in each data record and the order of the records in the file is based on this field. As a result, the order of the data records may differ substantially from the order a user might expect. The physical sequence, of course, influences the length of search and access time for data records in a sequentially organized data file, thus affecting the user's processing costs.

Missing data records that are lost due to processing errors (e.g., in tape copying) or are truly non-existent (because no tabulations exist for that Census tract) present another costly dilemma for the user. Such omissions are usually discovered as the data are processed, and thus may force the user to suspend or redesign his analyses in mid-stream. Unfortunately, preliminary completeness checks are also costly unless they can be performed as a by-product of tape copying (e.g., a list of all geographic areas represented on file).

ALTERNATIVES FOR DATA ACQUISITION

To the potential Census data user, the next step is obtaining access to the data he or she needs. In addition to the Census Bureau itself, there are several other types of agencies through which Census data are available. These agencies fall roughly into three categories: Census-designated processing centers, data suppliers, and suppliers of computer services.

Organizations which are designated by the Census Bureau as processing centers provide data retrieval and tape copying services for census data users. These centers may be private or public, and operate for profit or on a nonprofit basis. They are not controlled or certified by the Bureau of the Census, but once recognized they are added to a list of such centers which is available from the Bureau on request. Since each center is operated independently, a user may find from one tape reel to the entire Census available at the centers in his location.

Various private companies serve as intermediaries between the Census Bureau and Census data users. These organizations operate in a "value-added" mode, supplying users with data that is in a more efficient format than that provided by the Bureau. A most extensive variety of services, such as special tabulations and software products, also available from such agencies. Two organizations which supply Census data and services in this vein are the National Data Use and Access Laboratories of Rosslyn, Virginia (DUALabs), and the National Planning Data Corporation of Ithaca, New York.

Some organizations that market computer batch processing or time-sharing offer Census data services to their users. Such services include access to selected portions of the Census data and/or software access and display packages. Those services available over time-sharing networks, such as CENSAC on National CSS, Inc. and CENSTAT on CDC's CYBERNET, usually include data access plus software. Smaller services bureaus, such as UNI-COLL, Inc., Philadelphia, Pa., offer customized software systems to process data obtained by the users.

The advantages to using any of these secondary sources for Census data are ease of access and the availability of special services, e.g., consulting or software. However, ease of access is limited to those data made available by the organization selected. Many of these agencies concentrate on a few oft-requested tabulations or on one geographic region. Even those from which all data is available may charge more for access to data items that are rarely used. Further, the software services provided are usually standardized, with customized processing or special tabulations available at extra cost.

A user who wishes to obtain the original tabulations and do his own processing has the choice of dealing directly with the Census Bureau or with one of the secondary sources that provide tape copying services. The advantage of the latter is the receipt of data in a much more efficient form. For example, as offered by the Bureau of the Census, most Counts are supplied on a one state per tape reel basis—quite uneconomical in terms of storage and processing costs.

MANIPULATION OF THE CENSUS DATA BASE

The Census Bureau had maximum transferability as its objective in structuring the files of Summary Data and the Public Use Sample. In response to this objective, the data are made available in the most standardized fashion: in sequential organization, on magnetic tape, recorded at low density, encoded in one of two standard character codes (EBCDIC or BCD), often one state per reel, and with equal, fixed-length records. These characteristics, however, inflate the size of the Census data base and make its manipulation most inefficient. To improve storage and processing efficiency, the user should investigate changes in form and size, storage medium, and organization.[12]

The size of the Census data base is unnecessarily inflated by the use of equal-length data records and standard character codes. The former necessitates a large proportion of blank characters in many files, to fill fields unused in those records. The latter requires larger data fields for numeric values than would be necessary if binary representation were used. After making the above changes the user may be well-advised to pursue further reductions in data base size through the application of data compression techniques.[5] Zero suppression and simple character packing (two digits per byte) achieved compression ratios from 43 percent to 73 percent in the University of Pennsylvania Census data base.[12]

An appropriate change in storage medium, e.g., from magnetic tape to disk, must be determined with respect to the user's processing installation. The factors involved include the amount of data to be stored, the relative costs of tape versus disk processing, and the extent and frequency of processing anticipated.[13] In the University of Pennsylvania environment, transfer of user-requested tabulations from tape to disk as needed resulted in a sizable reduction in processing costs for a modest (and temporary) increase in storage charges.

Efficiency in accessing Census data can be further improved by changes in the organization of the data. Access to the Census data, as originally produced, is constrained by all the limitations of sequential files: variable access times for data dependent on location within the file, linear search for desired data, and a minimum number of data relationships actually represented by the file structure (e.g., geographic proximity or association by household). However, radical variation from the original organization puts the user and his data at variance with the published documentation. While this may be acceptable for a small portion of data, users maintaining a full Census data base, e.g., for a university community, would find the extra documentation burden unwieldy.

The approach to change in Census data organization at the University of Pennsylvania was to retain the original sequence of data records while regrouping these records into separate sub-files, and building an index to the new sub-files. Sub-files were defined either by geographic region (e.g., counties) or by record count (e.g., 500 records). In both cases, access costs for data in these files was substantially (50 to 60 percent) less than for access to the original state file. The level of indexing employed, however, still required sequential search within the sub-files and use of the index to locate specific records was limited by the sequencing imposed on the original file.

CONCLUSIONS AND RECOMMENDATIONS

The problems that face potential Census data users with respect to data description, acquisition, and manipulation indicate that the Census data base is not readily available to most researchers. With the next decennial Census rapidly approaching, it behooves those who have experienced some of these problems to suggest ways in which they might be ameliorated.

One recommendation would be that the Census Bureau assume the role of data base administrator (DBA) for the Census data base. That implies that while making the data available to a wide range of users, the Bureau would retain control over the data base. A radical means of doing this, that is now technically feasible, would be to make the Census data base available over a nation-wide network which users pay to access. In this way, the Bureau would physically retain control over the data and be able to assure their physical integrity and structural validity. Access methods and utility programs could then be standardized and shared amongst all Census users.

Another approach would have the Census Bureau retain only *logical* control over the data base. This would be an extension of the Bureau's current mode of operation. Portions of the Census data base would be physically transferred to users (for a fee) but the Bureau would be responsible for promoting accurate and efficient manipulation of Census data by applying the tools of documentation and standards.

To fulfill such a role, Census documentation would have to be improved in the following ways:

(a) It must be structured and cross-referenced as an integrated hierarchy, where the user could start at a general subject area and easily locate all relevant information down to the nature of specific tabulations and areas;

(b) it must be dynamic, so that continuing notices, error advices, and new data products could be easily reflected when they are pertinent; and

(c) it should be priced in such a way that participating research organizations and libraries would be motivated to purchase the entire set, not isolated fragments.

The concept of standards with respect to Census data would also have to be broadened. In addition to formatting standards required for transferability, the Bureau could provide standard access routines, oft-requested utility routines (e.g., for elementary statistics) or interface routines for the most common statistical packages, e.g., SPSS or BIOMED. For users with incompatible processing environments, these standards could be set forth in the form of guidelines or logic flowcharts, from which users could create their own software.

As with any other data base, once administrative tasks are removed from each group of diverse users and returned to a centralized agency of control, access and integrity will be improved. To do less with a data base of the size and importance of the Census, is to implicitly restrict public access to a valuable national resource.

ACKNOWLEDGMENT

The author wishes to express her appreciation to Mr. David F. Sanford, Wharton Econometric Forecasting Associates, Philadelphia, Pa., for his consultation and advice on this paper.

REFERENCES

1. *CENSAC Users Guide,* NCSS Form 954, National CSS, Inc., 300 Westport Avenue, Norwalk, Conn. 06851, January 1973.
2. *CENSTAT Information Manual,* Pub. No. D0001902501, CDC Data Services, December 1970.
3. *1970 Census Data Finder,* Clearinghouse and Laboratory for Census Data, Suite 900, 1601 North Kent Street, Rosslyn, Va. 22209.
4. *Data User News,* Subscriber Services Section (publications), Bureau of the Census, Washington, D.C. 20233.
5. Kreutzer, P. J., *Data Compression in Large Business-oriented Files,* Navy Fleet Material Support Office, AD-734394, October 5, 1971, NTIS, Springfield, Va. 22151.
6. National Data Use and Access Laboratories (DUALabs), 1601 North Kent Street, Rosslyn, Va. 22209.
7. National Planning Data Corporation, 20 Terrace Hill, Ithaca, New York 14850.
8. *UNI-COLL Bulletin,* "Census Information System," UNI-COLL Corporation, University City Science Center, Philadelphia, Pa., Summer 1973.
9. U.S. Bureau of Census, *1970 Census User's Guide—Part I,* Washington: Government Printing Office, 1970a.
10. U.S. Bureau of Census, *1970 Census User's Guide—Part II,* Washington: Government Printing Office, 1970b.
11. U.S. Bureau of Census, *Public Use Samples of Basic Records from the 1970 Census: Description and Technical Documentation,* Washington: Government Printing Office, 1972.
12. Weldon, Jay-Louise, *Data Storage Decisions for Large Data Bases,* Doctoral dissertation, University of Pennsylvania, August 1975a.
13. Weldon, Jay-Louise, "Implementation Strategies for the Census Data Base" (abstract), *Proceedings of the International Conference on Very Large Data Bases,* September 1975b.

Defining management's information needs

by TREVOR JOHN BENTLEY
Tilling Construction Services Ltd.
Collingham, England

ABSTRACT

Before we can discuss Management Information Systems on a coherent and beneficial level we must know the needs of those at whom the information is directed. It has been said that a good salesman can create needs for his product and there is no doubt that this is what has happened in the development of computer based systems. It is time we found out what our customers' real needs are and it is time that we admitted to ourselves that these needs may not always require the most sophisticated solution.

This paper describes a survey which sets out to establish management's information needs by focussing on the decisions taken by the managers and the information necessary to provide the input for the decision process. This identification of decision points and the subsequent analysis of information needs is an essential prerequisite for the successful design of meaningful information systems. The steps to be taken are set out in detail together with my comments based on the practical experience of carrying out such an analysis. At this time the survey has not been completed but by June '76 results should be available for discussion.

THE PROBLEM

The problem to which this research addresses itself was clearly defined by James D. Gallagher who sets the following goal for an information system.

> "The ultimate goal of an effective management information system is to keep all levels of management completely informed on all developments in the business which affect them. To do this, the data-processing personnel and those entering information into the system should know exactly what data to collect and which to tabulate, and management on its part has the obligation to be able to write down its actual requirements for internal information."[1]

The problem of knowing "exactly what data to collect and which to tabulate" is not new to management. Edward T. Elbourne recognized this in 1914 when he wrote.

> "It is quite possible for the Management to collect more information than it can use to advantage, or which is more costly, or hinders production more, than the information is worth. This is a real danger that has to be guarded against continuously, for routine that serves a valuable purpose when initiated may cease to be useful by some later change in conditions."[2]

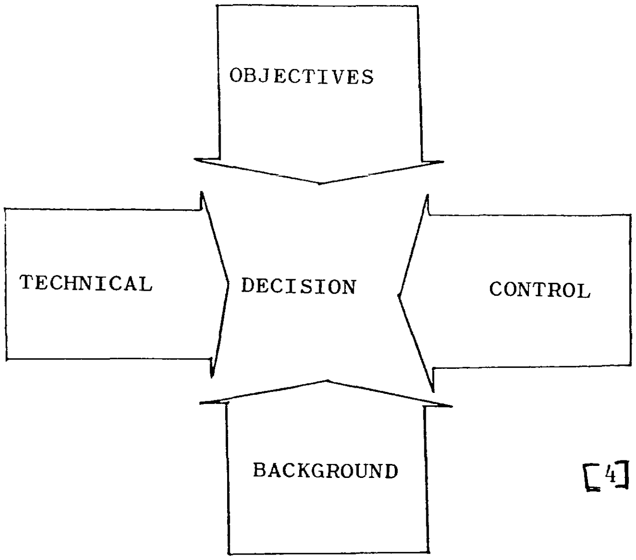
Information is the raw material which the manager needs to make a decision. Without information the manager is unable to carry out his function in the organization.

> "The manager needs information to assist him to select courses of action i.e., take decisions, to control the implementation of action and to record the success or failure of the action taken. It is necessary therefore to define the decision making areas of each manager's job in order to provide information which will be of help."[3]

The relationship of information to decisions is fundamental to the research and is clearly a vital consideration in the development of the right approach to the problem. The manager must then receive information related to his job, his responsibilities and the decisions which he takes. Such information can be broadly categorized.

"The manager needs several kinds of information:

(a) the objectives which he is to attempt to achieve;
(b) technical information about specific jobs, for which he is responsible;
(c) control information based on feedback of the results of decisions so that corrective action can be taken;
(d) background information about activities related to those for which he is responsible and of the company and the environment in which he is operating."



Some of this information will be available, some will not. It is obtained from both internal and external sources. Managers already have sources of information which they use for making decisions and the research must start from the present position and examine—

1. The information presently used,
2. The information needed but not available, and
3. The information which cannot be obtained.

> "The manager will never be able to get all the facts he should have. Most decisions have to be based on incomplete knowledge—either because the information is not available or it would cost too much in time and money to get it. To make a sound decision, it is not necessary to have all the facts; but it is necessary to know what information is lacking in order to judge how much of a risk the decision involves, as well as the degree of precision and rigidity that the proposed course of action can afford. For there is nothing more treacherous—or alas, more common—than the attempt to make precise decisions on the basis of coarse and incomplete information."[5]

The research will aim to answer the following questions.

1. How can key decisions be isolated?
2. Can a decision structure be devised?
3. How can the information requirements of the decisions be assessed?
4. Can the utility of the information be determined?
5. How can the degree of risk be related to the availability of information?
6. Can an information system be designed to satisfy management's needs?

Information is a peculiar thing which varies depending upon when it is received, how it is received and who receives it. The same piece of information may be interpreted differently by different managers depending upon their attitudes and approach to the decision concerned.

> "Like management itself, management information has vital human implications. . . . To demonstrate a point, then, let's consider the implications to various people of a train whistle penetrating the evening dusk.
>
> To the saboteur crouching in a culvert it might signify the failure of his mission because the whistle indicates that the train has already passed over his detonating charge without causing an explosion. To the playboy it might presage the imminent arrival of a transgressed husband. To the fireman in the cab of the locomotive it indicates a drop in steam pressure and the need for restoking the furnace. To the lonely wife it means the return of her travelling husband. To the man with his foot caught in the switch down the track it preshadows doom. . . . For another (preparing to retire) it signifies time for prayer. . . . In brief, the nature and significance of any information are fundamentally and primarily functions of the attitudes, situations, and relevant responsibilities with respect thereto of the people involved with it. . . .
>
> . . . Information is management information only to the extent to which the manager needs or wants it; and it is significant to him only in terms of its relation to his accumulation of relevant knowledge and plans and to his personal responsibility."[6]

The problem being faced is therefore a complex and difficult one. There is unlikely to be a specific solution, but if a way can be defined to analyze and categorize information needs then the aim will have been achieved.

> ". . . There is a frighteningly common desire today to prove that incredible amounts of information can be developed with electronic devices by preparing business reports that are incredibly long, incredibly dull, and, all in all, just plain incredible.
>
> Information alone is not enough. Try visualizing for example one of our big daily newspapers if it were presented straight off the wire in continuous columns, with no headlines, no attempt to avoid duplication, and no simple means of judging the relative importance of the various news stories or putting them in proper perspective. Would you even attempt to read such a paper? I think not. Yet management is frequently forced to hunt through a haystack of irrelevant information in its reports in order to find for itself the needle of pertinent fact. What is needed, obviously, is a planned system of business intelligence—or, as the

author of this report prefers to call it—a "management information system" which selects, rejects, edits, and headlines business information—in short, which turns it into business intelligence.[7]

THE SURVEY

Meaningful information

Meaningful information can be defined as follows:

(a) For data to be called information it must add to the manager's store of knowledge.

(b) To be meaningful the increase in knowledge must be relevant to the manager's decision-making activities.

For example there is no point whatsoever allocating overheads to cost centers when the cost center manager has no control over those costs. Neither control nor information will be improved by such arbitrary accounting conventions.

It must be realized that the majority of the data presented in monthly accounting reports does not fall into the category of 'meaningful information', as most of the data contained in the accounts is already known to the manager from whose activity it originated. It is certainly not new to him. It has simply been converted into misleading monetary terms. This is particularly so in a period of high inflation. It is quite clear that we must provide data in terms of quantities and hours and other inflation-proof measures.

Defining management's needs

It is vital that management's information needs are examined so that meaningful information systems can be designed. The System Designer, however, must not:

(a) Ask the manager what he wants; simply because he will not be able to answer, unless the question can be related to his decision areas.

(b) Tell the manager what he needs; this will and does cause resentment and leads to systems oriented managers rather than management oriented systems designers.

(c) Give the manager what is available; this is the most common practice and has, I believe, led to the paperwork explosion that is burying management in useless data. The computer has unfortunately added to the paperwork explosion by providing more information which is now more readily available, a good deal of which is irrelevant.

The systems designer and the manager working together must establish the following.

(a) The decisions taken by the manager.

(b) The information ideally required for those decisions.

(c) The information currently available.

It is probable that this analysis will result in the situation depicted in Figure 1.

Achieving the above is more difficult than it might appear, and unless a carefully structured approach is taken it will be impossible. There is no alternative to a detailed systematic analysis on the following lines.

Step 1: *Determine decisions*—This can only be done by spending some time with the manager and learning from observation and analysis the decisions he takes. It will be apparent from this that his decisions will fall into the following categories.

Routine: taken regularly; highly structured with easy access to the data required, e.g., raising a credit note for a pricing error.

Mechanical: taken less frequently, but still structured with known data requirement, e.g., producing a production schedule.

Complex: taken infrequently, unstructured, depending largely on current circumstances. Unknown information requirement, i.e., cannot be predetermined.

If the impact of these decisions on the managers' results can be assessed, then a Decision Grid can be completed for each manager. (See Figure 2)

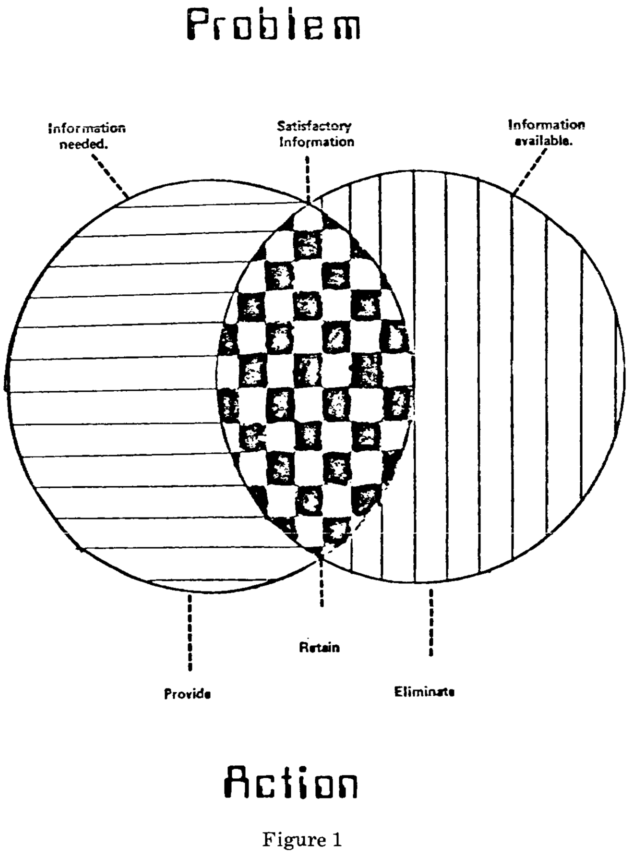


Figure 1

KEY DECISION GRID

IMPACT SPECTRUM	Minor	Important	Vital
COMPLEX			
MECHANICAL			
ROUTINE			

Figure 2

Step 2: *Information Analysis*—For each of the decisions on the Decison Grid it is necessary to produce an Information Analysis Grid (See Figure 3), which records the information ideally required in the following sections.

Class: Available and used
Available not used
Not currently available

and

Category: Vital
Desirable—economic
Desirable—uneconomic

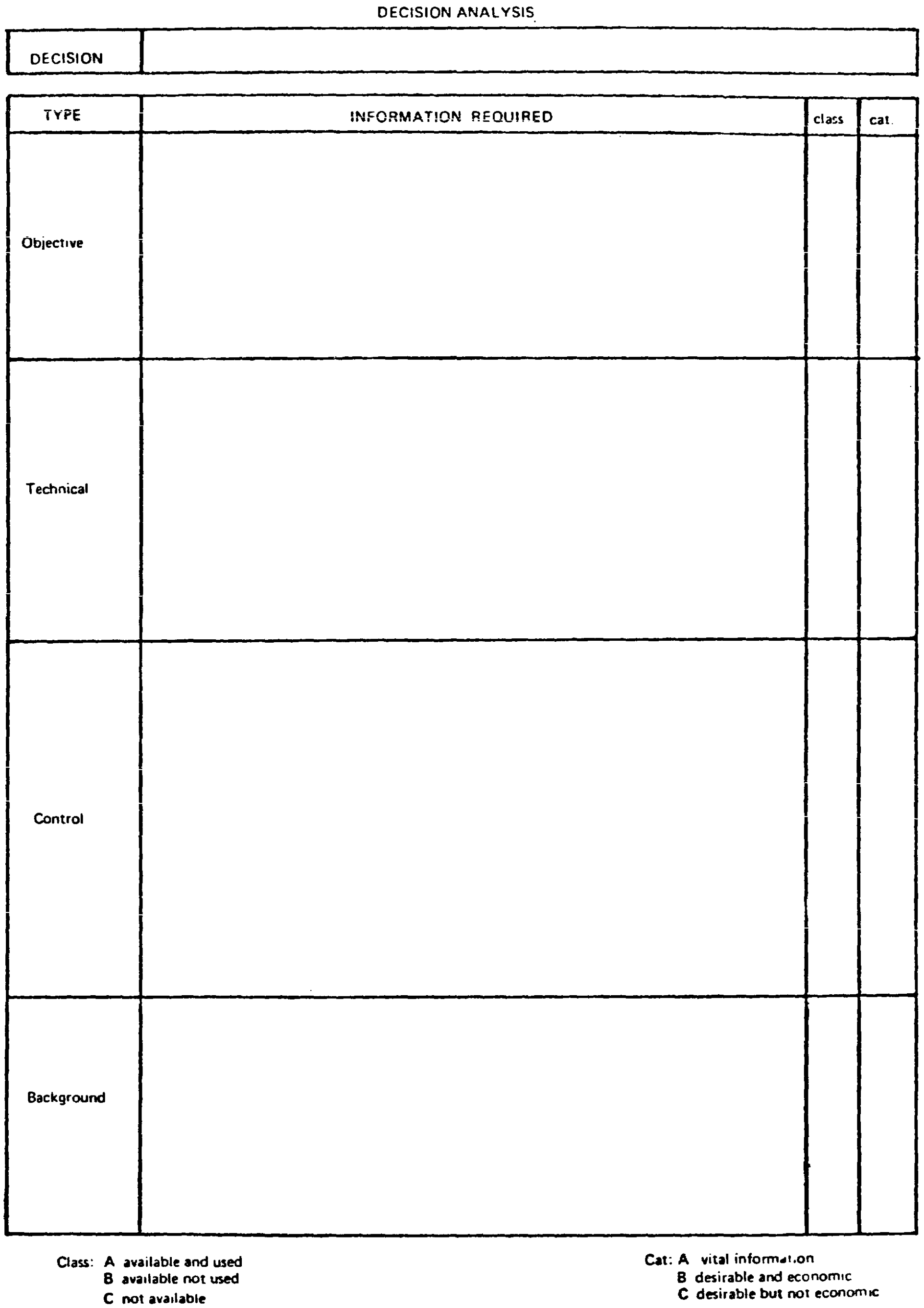
It is then important for the manager to assess the degree of risk if the information is not available. This can be recorded as HIGH, LOW, or by the use of probability scales.

Step 3: *Decision Analysis*—With the above information it is now possible to complete the Decision Analysis

INFORMATION ANALYSIS GRID

	VITAL	DESIRABLE		Degree of RISK if information is not available
		economic	uneconomic	
Available and used				
Available not used				
Not currently available				

Figure 3



DECISION ANALYSIS

DECISION	

TYPE	INFORMATION REQUIRED	class	cat.
Objective			
Technical			
Control			
Background			

Class: A available and used
B available not used
C not available

Cat: A vital information
B desirable and economic
C desirable but not economic

Figure 4

(See Figure 4) which is the master document for the design of the system. On the Decision Analysis the information requirement is split into the four main types of information—

Objectives (Plans)
Technical
Control
Background

This will be used to define where the information comes from, the frequency, content, accuracy level, etc. All important points for effective systems design.

Step 4: *Providing the Information Required*—Referring to Figure 1, Action should be taken to—

(a) produce the information required,
(b) eliminate the information not required.

Producing the required information may take some time to achieve as it will call for the amendment of some systems and almost inevitably the re-design of other systems. A good deal of useless information can be dropped fairly quickly and this should provide some relief for the inundated manager.

The above steps are implemented in the following way.

(a) The managers concerned are invited to a seminar which explains the relationship of information to decision making and its importance to their individual performance.
(b) Following the seminars the managers are visited and interviewed by the researcher who assists in the completion of the Decision Grid and Analysis Sheets.
(c) The researcher then analyzes the results and presents a report covering:
    (i) The decision points.
    (ii) The importance of each decision.
    (iii) The information required.
    (iv) The existing source, if any.
    (v) The basis of the new system if one is required to provide information not currently available.

Throughout the above the researcher must act only as an adviser and analyst, it is imperative that the manager assesses his own information needs related to the decisions which he takes.

BASIC INFORMATION SYSTEM CONCEPTS

When the manager's needs have been established the kind of Management Information system which is most suitable for the circumstances must be examined. It is fatuous to think that the existing system is of any use whatsoever until the comparison between what is needed and what is available has been made.

There are three principal categories of Management Information Systems.

(a) *Data Bank*
All data is recorded for every transaction and placed on file. It is then available for answering questions.
(b) *Systems Basis*
Information flow designed to accept and process data as a management tool. Leads to extensive systems with complex programming.
(c) *Combined Systems/Data Bank*
This is a Decisions Based System approach and provides systems for certain areas oriented to decision making and a data bank for others dependent entirely on management needs.

The most applicable approach for the majority of companies is that indicated in (c) above, namely a combined systems/data bank approach, the reasons for this are as follows:

(a) Complete data banks are impracticable, in that much of the information is obsolete before it reaches the file and even when it is there it can be very difficult to retrieve. Questions cannot be sufficiently pre-determined to allow for an adequate questioning sequence to be built into the file. Some of the underlying concepts of this approach are not acceptable e.g., all data is not valuable and generally it depreciates with time. In addition to which the form content and frequency of information should not be dictated by the systems, but by the needs of the decision making process.
(b) The total system approach requires extremely complex systems designed on the basis of how information can be used by functions and the understanding of the flow between functions and processes. This makes the whole information network system oriented, inflexible and too complex to be understood by the management using the information.

Decision based systems have several advantages.

(a) It is necessary to identify decision areas and then ensure that resources are available.
(b) Management's information needs must be clearly defined and it is usually established that less but more relevant information is required.
(c) Information is directly related to the task and thus ensures it is available in the right quantity, of the right quality, and at the right time.
(d) More efficient use of computer hardware is possible when it is directed towards management needs.
(e) The production of relevant information for making decisions should lead management, if they use this information correctly, towards making better decisions.

COMPUTERS AND MANAGEMENT INFORMATION SYSTEMS

The term Management Information System is linked in most people's minds inexorably with computers. This link is understandable as all current literature on Management Information Systems concerns itself exclusively with computer based systems. It is not, of course, necessary to use computers when discussing information systems, they do have their place and it is part of the System Designer's function to recognize when to use computers and when not to use computers.

The computer's main strength lies in three areas.

(a) Routine Data Processing.

(b) High capacity fast access storage and retrieval facilities.
(c) Mathematical models for simulation.

Routine data processing is mainly concerned with accounting procedures and related analysis for information purposes. This is still the major application area, particularly in the U.K. The computer's ability to handle large volumes of routine data extremely quickly in batch mode has led to the wide development of such routine batch processing applications. They are not as glamorous as the so called "Integrated Management Information System" applications few of which exist.

The use of high capacity, fast storage and retrieval facilities has led to the growth of on-line and real-time operating systems, which however effective at controlling airline bookings, do not provide much if any management information. A large UK holiday company used real-time systems for booking control, and then batch processed the data for accounting and information purposes. Unfortunately the company went bust.

Simulation models are undoubtedly a valuable management tool and can aid decision making. However, their development requires a high degree of mathematical competence, and a profound understanding of the business problems. Attributes rarely possessed by the same manager. If models are to be used effectively then they must be—

(a) Small and relatively simple,
(b) Used regularly, and
(c) Built by managers.

One of the risks of using computers in decision making is that the model builder will attempt to construct programs containing value judgments, and it is here where failure must occur. In addition the social aspects of decision making cannot be programmed.

> "Since no computer programs have yet been written which pick from an open-ended range of possible selections, it is now impossible to arrive at 'managerial' decisions by automatic process."[8]

The system designer must first assess the problems before he attempts to develop any single approach to the solution. I personally believe that the computer specialists have for too long been offering management a solution in search of a problem.

It would be of great benefit to most companies if the systems designer obtained answers to the following questions before computers were included in the plans.

(a) Is the existing non-computer system the best possible?
(b) Is there any other way?
(c) Do the system requirements fall within the strengths of the computer?
(d) Five reasons why a computer must be used over any other system.
(e) The importance of the system to the company.
(f) What will the company lose if the system fails?
(g) What are the benefits in £p from successful implementation?

STAGES IN DEVELOPING THE MANAGEMENT INFORMATION SYSTEM

As we have already seen the systems designer is faced with three major problems in designing a decision based information system.

(a) Obtaining an understanding of information and its importance to the managerial function.
(b) How to establish what information any particular manager requires in order to meet his decision making needs.
(c) The means by which this information can be collected, stored, processed and retrieved.

At the beginning of this paper I suggested a means of establishing management's information needs. The systems designer, by using the approach indicated, should have obtained answers to the following questions.

(a) What are the key decision areas at each level of activity?
(b) What information is required to make the decisions?
(c) What information is lacking?
(d) How can it be obtained and at what cost?
(e) Does it require amending existing systems or introducing new ones?

A systems framework can be established which will indicate what work has to be done to provide the needs of management. Existing systems cannot be withdrawn and replaced overnight, so a plan has to be formulated. Rationalizing existing systems based on this framework will produce the most immediate benefits. This is done in two stages:

(a) Prepare improvement program indicating the priority areas.
(b) Simplify existing procedures.

The flow of information from computer based systems must be examined to ensure that the files hold data in the most useful form for meeting the information requirements of management. The conflict arises as follows and limits the effective use of the computer.

(a) Files are designed to hold information in the sequence most appropriate to the operation of the computer systems.
(b) Information required is seldom needed in the same sequence that facilitates rapid operation on the computer.

The possible solution is to create files based on information needs and secondary files or improved systems to handle the operational data needs.

The provision of an information system for management must be tackled slowly with the development of individual subsystems linked together by the decision based reporting system.

Developing new systems should be one of evolution for the following reasons.

(a) Too many activities to be absorbed at one time.
(b) Human effects of change.
(c) Complexity of changes.
(d) Limitation of available man hours for effective systems design.

CONCLUSION

I am convinced that the provision of—

the right data, at
the right time, in
the right place, for
the right reasons,

is the principal aim of the systems designer, however, he cannot achieve this aim without a close involvement with management and a deep understanding of management decision making processes.

REFERENCES

1. Gallagher, James D., *Management Information Systems and the Computer*, American Management Association, New York 1961.
2. Elbourne, Edward T., *Factory Administration and Accounts*, The Library Press, 1914.
3. Bentley, Trevor J., "Designing a Management Information System," *Conference Proceedings*, O & M International, 1975.
4. Bentley, Trevor J., "Designing an Information System That Meets Management's Needs," *Management Accounting*, November 1974.
5. Drucker, Peter F., *The Practice of Management.*
6. Dwyer, Edward D., "Some Observations on Management Information Systems," *In Advance in EDP and Information Systems*, American Management Association, New York, 1961.
7. Phillippe, Gerald L., "What Management Really Wants from Data Processing, *Data Processing Today, A Progress Report*, Management Report No. 45, American Management Association, New York, 1960.
8. Jones, Curtis H., *Harvard Business Review*, September/October 1970.

Managerial response to an information system

by ROY H. IGERSHEIM
American Management Systems Inc.
Troy, Michigan

ABSTRACT

One of the most common problems in implementing a successful information system is its threatening nature to users of the system—particularly middle managers. The behavioral implications inherent in the implementation of an information system were studied by testing the following two propositions:

- Acceptance of an information system is positively related to involvement in the implementation of the information system.
- Acceptance of an information system is negatively related to the perception of the system as threatening.

Three hundred thirty one middle managers from five different organizations were sampled. An overall response rate of 72 percent was achieved utilizing an anonymous three-part questionnaire. Various statistical techniques were utilized to validate the proposed scales used in testing the hypotheses.

Based upon the data analysis, the two propositions are strongly supported. There is a definite positive relationship between a middle manager's acceptance of an information system and his participation in the analysis and design of the system. In addition, there is a negative relationship between a manager's acceptance of an information system and the perceived threat of the system to such behavioral factors as job satisfaction, job skill, job opportunity, job originality, job status, and job salary.

INTRODUCTION

The entire management information system (MIS) field must still be considered to be in its embryonic stage. For approximately the last two decades computer-based information systems have been utilized by industry and government and the computer's potential recognized as a vital management tool in daily operations. This increased capability has brought about complex data processing (D.P.) applications and a growing awareness of the need to integrate these applications in the form of a MIS. In turn, this has led to numerous technical and personnel problems that must be overcome if such systems are to be successful.

One of the most common problems in implementing a successful information system has been its threatening nature to users of the system—particularly middle managers. It is a widely accepted tenet that information systems should be designed, developed, and implemented for middle managers. Generally, however, their needs and perceptions about the system have been given little, if any, consideration. If these middle managers fail to understand or accept the system, it is not likely that they will use it to perform their job more effectively.

The behavioral implications inherent in the implementation of an information system were, therefore, studied. Specifically, the study determined whether or not middle managers in five selected organizations perceive the MIS as threatening in terms of various behavioral criteria factors and whether or not they accept the MIS. The following two propositions were primarily examined:

1. Acceptance of an information system is positively related to involvement in the implementation of the information system.

2. Acceptance of an information system is negatively related to the perception of the system as threatening. The following behavioral variables were examined in this regard: job satisfaction, job skill, job opportunity, job originality, job status, and job salary. The interrelationships of these variables were also examined; therefore, as a corollary to the two primary propositions, it was proposed that a positive relationship exists between each of the behavioral variables.

It is surprising that the behavioral problems associated with information systems have not previously been investigated. Much has been written describing the characteristics of a successful management information system. However, the main emphasis has been in the direction of technical factors—hardware and software—and overall MIS considerations such as problem definition, problem analysis, and problem solving. Research in the area of user response to the MIS and how acceptance/lack of acceptance is related to their involvement in the implementation of the MIS and the perceived threats that the MIS has to them has not yet been undertaken. There has also been extensive research in the area of attitude measurement,[9,12,18,25] but little of this area pertains to information systems. Instead, research has disclosed that there are few operational computerized management information systems considered to be successful—possibly because the human factor was not given enough consideration. As Allen Rowe observes, people are the key ingredient in a MIS.[19] M. Scott Myers actually defines a MIS as a "process of people interacting in order to apply resources for the achievement of various goals."[16] Neither of these authors, however, nor any others present any empirical evidence regarding the behavioral implications of a management information system.

It is also necessary to note that during the past decade numerous terms were used to describe the different types of computer and systems applications that were being developed. The same term has been defined in different ways by various authors and in different disciplines.[4,7,21,23] For this reason and to avoid any possible misinterpretation the following meanings are to be given to the following terms.

A. Management Information System—or Information System for Managers—a system that provides the proper information to the proper person, at the proper time and at the proper cost.

B. Middle Managers—composed of those members of management above the level of foreman and below the level of vice president.

C. Perceived threat—the perceived potential forced movement away from a desired position or date.

METHODOLOGY

The sample consists of 225 middle managers from five organizations possessing an actively-utilized, computer-based management information system. Two of these five organizations are agencies of the United States Government based in the Washington, D.C. metropolitan area; the other three are large eastern industrial corporations. Middle managers were selected as subjects because they are the prime users of the MIS. In the governmental agencies, this group was comprised of GS12's, GS13's, GS14's and selected GS15's. A breakdown of subjects by organization can be found in Table I. A total of seventy-two percent (246 respondents) of the questionnaires distributed (341) were returned and found usable.

Data was collected from July through October 1973. All participants completed a three-part questionnaire developed for this study. Subjects were in no way asked to identify themselves—either by name or job function. This questionnaire required the respondents to react along a six-point continuum to 57 attitudinal statements concerning the MIS and to six statements measuring their involvement in its implementation process. The statements asked and the variables being measured were arrived at based upon a review of the MIS and behavioral science literature and the author's prior experience in the field.

Top management distributed the questionnaires with a cover letter as subjects were from several different departments and often different physical locations. Also, the subjects in this study within the industrial organizations worked on different shifts and previously started vacations of the subjects could not be rearranged. The direct support of top management and anonymous nature of the questionnaire assured a greater response rate. At least one follow-up letter was sent to all participants in this study approximately one week after the initial distribution of the questionnaire. In two organizations, questionnaires were returned anonymously to top management and then forwarded to the author; in the other three organizations, subjects were furnished stamped, self-addressed envelopes and questionnaires returned directly to a Box number. As can be observed from Table I, the response rate was not affected by the different methods of questionnaires return.

Factor analysis[13,15,24] was used to test the *a priori* scale structure. The purpose of this analysis was to reduce the number of variables to those showing a common variance and to determine if the items making up each *a priori* scale were in fact parallel. The *a priori* structures did generally, in fact, hold up.

The *a priori* scales were then modified as a result of the factor analysis and these modified scales were then used.

The mean score and standard deviation of each of

TABLE I*

	Total Subjects Returning Questionnaires	Total Questionnaires** Distributed Within Organization	Response Rate In %	Total Usable Questionnaires
Organization 1	59	86	69	54
Organization 2	55	83	66	51
Organization 3	33	44	75	30
Organization 4	57	73	78	49
Organization 5	42	55	76	41
Total	246	341	72	225

* The participating organizations wished to remain anonymous; therefore, no organization names will be given. Organizations 1 and 2 are governmental agencies; organizations 3, 4, and 5 are industrial firms.

** The distribution of questionnaires within each organization included all middle manager/users of the management information system within one or more specific departments.

the scales was computed and then the scales were correlated using the Pearson Product Moment Correlation in the SPSS statistical package.[17] The scales were correlated on an individual basis by organization and then these organizations were combined and grouped together as a composite (Table II).

RESULTS

The results of this study are presented in Table II: Columns 1 through 5, for each behavioral factor, contain the correlation coefficient of one individual organization. Column 6 for each behavioral factor contains the correlation coefficient for the entire sample. As can be observed the scale reliability using the Veldman test (α) for each of the factors is greater than .632 which appears to indicate that the resultant scales, based upon the factor analysis, are quite reliable. In fact, the reliabilities of the involvement, acceptance, job satisfaction and job originality scales are greater than .8.

In addition, many of the correlation coefficients have a significance level of greater than .001. This indicates that the proposed relationships for each individual organization and for the composite grouping of organizations strongly support the basic propositions. For example, the correlation coefficient between perceived job satisfaction and acceptance of an information system for all 225 respondents is .72. Given that the significance level of .001 is .020, this result appears to be quite significant. This implies that there is a strong positive relationship between acceptance of an information system and job satisfaction.

The data should be interpreted by using the following procedure. A high positive correlation statistic on the behavioral factors—job satisfaction, job skill, job opportunity, job originality, job status, and job salary—implies that there is low threat to those factors while a high negative statistic implies that there is a high threat to those factors.

DISCUSSION

To describe and analyze the data presented in Table II, this section examines each of the factors, determines which relationships exist, why they exist, and whether or not any pattern can be observed. In order to determine relationships between scales, each of the hypotheses will first be analyzed by individual organizations and then in composite form. Thus, the individual organizations participating in the survey can compare their data to other organizations and to the composite grouping.

Organization 1

The data for this organization shows that a strong positive relationship exists between acceptance of an information system and a manager's involvement in the implementation of that system (.36). This occurs possibly because the information system was designed by this organization's management and by a group of external consultants. This system was designed on a large scale third generation equipment, taking into account all of the users of the system. With regard to a manager's acceptance of an information system and a manager's perceived behavioral threats, a very strong positive relationship exists between acceptance and job satisfaction (.67), job skill (.64), job originality (.62), and job status (.60). In addition a strong positive relationship exists with the other behavioral

TABLE II

CORRELATION COEFFICIENTS FOR ALL INDIVIDUAL ORGANIZATIONS AND FOR THE COMPOSITE **

	ACCEPTANCE						JOB SATISFACTION						JOB SKILL						JOB OPPORTUNITY						JOB ORIGINALITY						JOB STATUS						JOB SALARY						INVOLVEMENT					
	1	2	3	4	5	6	1	2	3	4	5	5	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6
ACCEPTANCE α = .833*																																																
JOB SATISFACTION α = .827	.67	33	.76	.24	.53	72																																										
JOB SKILL α = 795	.64	.86	.71	.45	53	73	72	.92	.60	.57	.75	81																																				
JOB OPPORTUNITY α = 729	.30	.62	.47	.11	.40	.42	67	.81	.63	.47	.72	68	38	.76	.51	.42	65	53																														
JOB ORIGINALITY α = 810	.62	.80	.76	.25	59	.70	.87	.91	.34	.70	.88	.87	.80	92	77	.79	76	.84	68	.82	.64	.47	65	.66																								
JOB STATUS α = 645	50	83	.43	.43	.28	.63	.68	.86	.43	.41	60	70	.59	.86	.33	.26	.55	.68	.42	70	.30	.37	.62	.48	72	83	.66	.40	.59	.71																		
JOB SALARY α = .632	.35	59	.49	-33	06	.35	59	.74	.50	.23	45	.57	.34	72	.49	.13	.23	45	55	.78	57	.64	.70	.67	.55	.77	.55	.27	.24	53	.35	.70	.07	.14	.32	.39												
INVOLVEMENT α = 878	36	.55	.44	.39	34	.43	.40	.48	.31	.10	39	.39	.30	.55	.54	.35	25	41	36	.49	.36	.22	.50	39	38	.60	.48	.25	.41	44	.26	.49	.04	.01	.39	.31	.05	.42	.33	-.13	.30	.24						

* The α coefficient equates to the reliability of the scale using the Veldman test This test is based upon the entire sample of 225 cases.

** Column 1 thru 5 for each scale contains the correlation coefficient of one individual organization.

Column 1 is data based upon 54 cases & its significance level is {.001 > .42, .005 > .35}

Column 2 is data based upon 51 cases & its significance level is {.001 > .42, .005 > .36}

Column 3 is data based upon 49 cases & its significance level is {.001 > .41, .005 > .36}

Column 4 is data based upon 30 cases & its significance level is {.001 > .52, .005 > .46}

Column 5 is data based upon 41 cases & its significance level is {.001 > .44, .005 > .38}

Column 6 for each scale contains the correlation coefficient of the entire sample grouped together & its significance level based on 225 cases is {.001 > .020, .005 > .017}

Columns 1 & 2 contain data from two large governmental agencies.

Columns 3, 4, 5 contain data from three large industrial firms.

criteria and acceptance—job opportunity (.30), job salary (.35) and involvement (.36)—but they are not as strong as the former scales. This indicates that job opportunity, job salary, and involvement are not perceived as important as the other factors as it relates to acceptance. The data, therefore, appears to indicate that those individuals accepting the information system do not in any way feel threatened by it. Individual middle managers within this organization are very satisfied with their job, perceive that they have a high skill level, possess much job originality, and have a high degree of job status.

Upon examining the interrelationships of the various behavioral factors, generally one finds a strong positive relationship. The only major exception is the interrelationship of job salary to involvement (.05) which is not significant.

Organization 2

The data for this organization, a large governmental agency based in the Washington, D.C. area, indicates that all the proposed relationships are true. This organization has large third generation IBM computing equipment on which their information system was implemented in 1970. Primarily this information system is an output reporting system designed by external consultants with the advice of this organization's top management.

There appears to be a very strong positive relationship between acceptance of an information system by managers using the system, their involvement in it (.55); and acceptance of the information system and the lack of threat of the system. Individuals within this organization do not find the information system threatening. For example, the relationship between job satisfaction and acceptance (.83) is very strong in a positive way when one considers that a significance level of .001 is .42. The corollary propositions—the relationships between the behavioral variables—is also very strong. The data for this organization in comparing it with the other organizations is far greater than those in significance level and strength. All this appears to indicate that the individuals in this organization are not threatened in any way and accept the information system. This fact could be the result of employment by the government from which one's job is quite secure.

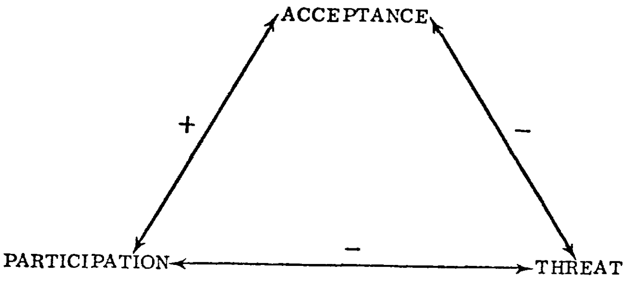


Figure 1—Visual results of hypotheses

Organization 3

The data for this large industrial firm is quite similar to the data for organization 2. This organization primarily uses its information system in the evaluation of various centers. As a consequence, therefore, it was designed by accountants for accountants to meet their needs. This small IBM computer system is used also to keep track of inventories and for warehousing purposes.

A very strong positive relationship exists between acceptance of an information system and involvement and acceptance and all the behavioral factors. It appears that the three most important factors relating to acceptance are job satisfaction (.76), job skill (.71), and job originality (.76). The other factors, though significant, are not as important. A conclusion that could be reached is the more satisfied a person is with his job, the more likely he is to accept the information system.

Among the interrelationships of the behavioral factors, job status appears to be less significant than the others. For example, the data appears to indicate that there is no relationship between job status and job salary or job status and involvement. This appears to indicate that salary and the amount of involvement in the information system by the manager has no bearing at all on his job status. This conclusion is quite plausible as the information system in this organization was primarily designed by the controller and his staff and many users of the information system were not involved in its analysis and design.

Organization 4

This organization, headquartered in a small city with rural plant sites, has an IBM computer system upon which the information system has been implemented. Managers in this organization really did not participate in the implementation process, but were forced to use the outputs of the system. Also many of these managers are located at various plants within a fifty mile radius of headquarters.

The data for this organization differs quite appreciably from the data from the other organizations. None of the propositions are strongly supported. In fact, acceptance of an information system is negatively related to perceived job salary (–.33). This implies that an individual's acceptance of an information system is threatening to his perceived salary level. Interpretation of this fact could lead to the conclusion that salary level is perceived to be lowered because of the information system. In addition, there appears to be no significant relationship between acceptance and job opportunity. Maybe the people in this organization do not feel that the information system gives them many additional opportunities possibly because of the location of this company and/or the backgrounds of the individual participants.

The remainder of the data generally supports the positive relationship purported. It does not appear to be as conclusive as the other organizations because the significance level is .005. Some of the interrelationships of the various behavioral factors are very low. For example, the following factors are quite low—job skill to job salary (.13), job status to job salary (.14), involvement to job status (−.01), job salary to involvement (−.19).

Organization 5

The data for this organization in general supports the propositions undertaken in this study. The relationship between acceptance of an information system and a manager's involvement in the implementation process is positive but less than the .005 significance level (.34). This statistic indicates that involvement is not as important a factor relating to acceptance as some of the other factors. The relationship between acceptance of an information system and an individual's perceived job satisfaction (.53), job skill (.53), and job originality (.59) is positive and very strong indicating that these factors are more important to an individual's acceptance of an information system.

The relationship between acceptance of an information system and job opportunity (.40) and job status (.28) is positive, but not as strong as the previous relationships. There is no appreciable relationship positive or negative (.06) between acceptance of an information system and perceived job salary. No possible explanation for this relationship could be found. The various interrelationships of the behavioral factors are all positive and generally they are quite strong. In fact the relationship between job satisfaction and job originality (.88) is extremely strong.

The composite

Upon examining the composite data (Column 6, Table II) one can observe that a very strong positive relationship exists between acceptance of an information system and the amount a manager was involved in its implementation and acceptance of an information system as it relates to each of the behavioral factors. It appears, however, that the data falls into two groupings:

—acceptance of the information system with job satisfaction (.72), job skill (.73), job originality (.70) and job status (.63).

—acceptance of an information system with job opportunity (.42), job salary (.35), and amount of involvement (.43).

Both of these groupings have a greater than .001 significance level but the first one ranges from .63 to .73 while the second one ranges from .35 to .43. This data implies that the key factors relating to acceptance of an information system are perceived job satisfaction, perceived job skill, perceived job originality, and perceived job status. What appears to be a significant finding is that involvement in the information system is not as important a factor as those above. Also, these composite findings, in general, support the findings of each organization.

In addition, the correlational data presenting the interrelationships of the various behavioral factors are all positive, have a significance level of greater than .001 and with the exception of involvement, are very strong. The following relationships appear to be the strongest: job satisfaction with job originality (.87) and job satisfaction with job skill (.81).

CONCLUSION

Based upon the data analysis, the two propositions appear to be strongly supported. There is a definite positive relationship between a middle manager's acceptance of an information system and his participation in the analysis and design of that system. In addition there is a negative relationship between a manager's acceptance of an information system and the perceived threat of the system to such behavioral factors as job satisfaction, job skill, job opportunity, job originality, job status, and job salary. The data presented in Table II clearly indicates that a manager who accepts the information system generally was involved in its analysis, design, or implementation and does not feel threatened by the system. Another way to demonstrate this conclusion is in Figure 1. This figure shows the positive relationship that exists between acceptance and participation and the negative relationship that exists between acceptance and threat and participation and threat.

Based upon the data presented it appears that in order to implement a successful information system the needs of the users (middle managers) must be taken into consideration. This can best be accomplished by getting them involved in the implementation process. If they participate in the design of the MIS, they will be more likely to accept it and use it as an aid in the decision making process. This fact should be considered by all systems analysts when designing information systems.

REFERENCES

1. Avots, Ivan, "The MIS Mystique, How to Control It," *Management Review*, October, 1970.

2. Canning, Richard G., "Trends in Data Management," *EDP Analyzer*, May, 1971.
3. Cougar, J. Daniel, "Seven Inhibitors to a Successful MIS," *Systems and Procedures Journal*, January/February, 1968.
4. Dearden, John, "The Myth of Real-Time Information," *Harvard Business Review*, May/June, 1966.
5. ———, "MIS is a Mirage," *Harvard Business Review*, January/February, 1972.
6. ———, McFarlan and W. Zani, *Managing Computer Based Information Systems*, Homewood, Illinois; Richard D. Irwin, Inc., 1971.
7. Edelman, Franz, "The Manager Looks at MIS," *Computer Decisions*, August, 1971.
8. Field, Roger, "Bringing the Universal MIS Down to Earth," *Computer Decisions*, June, 1971.
9. Fishbein, M., *Readings in Attitude Theory and Measurement*, New York, John Wiley and Sons, Inc., 1967.
10. Hanold, Terrance, "An Executive View of MIS," *Datamation*, November, 1972.
11. Head, Robert, "The Exclusive MIS," *Datamation*, September 1, 1972.
12. Johnson, Richard A., Fremont Kast and James Rosenzwerg, *The Theory and Management of Systems*, New York, McGraw-Hill Book Company, 1968.
13. Kerlinger, Fred N., *Foundations of Behavioral Research*, New York, Holt, Rinehart, and Winston, 1964.
14. Kriebel, Charles, "The Future MIS," *Business Automation*, June, 1972.
15. Likert, Rensis, *The Human Organization*, New York, McGraw-Hill Book Company, 1967.
16. Myers, M. Scott, "The Human Factor in Management Systems," *Journal of Systems Management*, November, 1971.
17. Nie, Norman, Dan Brent and C. Hadlai Hall, *Statistical Package for the Social Sciences*, New York, McGraw-Hill Book Company, 1970.
18. Porter, Lyman and Edward Lawler, *Managerial Attitudes and Performance*, Homewood, Ill., Richard D. Irwin, 1968.
19. Rowe, Allen S., "Coming to Terms with Computer Management," *Financial Executive*, April, 1968.
20. Schwartz, M. Herbert, "MIS Planning," *Datamation*, September 1, 1970.
21. Siegel, Paul, "MIS vs. EDP," *Modern Data*, June, 1970.
22. Spaulding, Asa T., "Is the Total Systems Concept Practical?" *Systems and Procedures Journal*, January/February, 1964.
23. Stern, Harry, "Management Information System—What is It and Why?" *Management Science*, October, 1970.
24. Thurstone, Louis L., *Multiple Factor Analysis*, Chicago, University of Chicago Press, 1947.
25. Trandis, Harry C., *Attitude and Attitude Change*, New York, John Wiley and Sons, Inc., 1971.

Transaction queuing and cylinder logic access in the Time, Inc. magazine/book/record system

by CARL R. GERAMI
Time, Inc.
Chicago, Illinois

and

T. RUSSELL SHIELDS and RICHARD J. WEILAND
SEI Computer Services
Chicago, Illinois

ABSTRACT

BRGE, the Time, Inc. Magazine/Book/Record online computer system manages one of the largest existing data bases directly updated online (five billion characters). System activity is managed by an extended CICS system with the ability to route and reroute transactions to appropriate terminals. A conglomerate transaction journal is maintained to serve as an audit trail and as the primary backup mechanism for restarts and recovery. When file restoration is necessary, the journal is simply used as a transaction source, and restoration is concurrent with continuing data entry. The data base is maintained on twenty-five 3330-II disks using the Cylinder Lists of Data (CLOD) file organization method, and accessed via the Cylinder Logic Access Method (CLAM). These permit both sequential and random access to a file, and handle overflow in a monolithic and extremely speedy manner.

INTRODUCTION

The Time, Inc. Magazine/Book/Record system (BRGE) is a CICS-based system running on an IBM 370/168 MP mainframe with 6,144K (6 megabytes) of main storage. The principal extension to CICS is the replacement of vendor-supplied file access methods with new file organization and access methods that give greatly improved access times and considerable additional maintenance flexibility and data base integrity. The system provides subscription fulfillment and related services for a customer community of 26 million subscribers.

About one-half of the customers are magazine subscribers to *Time, Sports Illustrated, Money, People,* and *Fortune.* The remainder are subscribers to book series (*The Old West, Life Science Library,* etc.) and record series (*The Story of Great Music, The Swing Era,* etc.) offered by Time. About two-thirds of system activity surrounds books and records, where customer activity tends to be roughly bimonthly rather than roughly annual as it is for magazines.

The system processes all transactions relevant to fulfillment, about 750,000 transactions per week. The transactions come from an optical scanner reading machine printed documents, and from 200 CRT terminals housed at the Time and Life Building in Chicago. BRGE handles incoming orders, payments, account adjustments, address changes, and provides a facility for customer services inquiries. In addition to the customer data base which will be described in some detail below, BRGE is also responsible for maintenance of ancillary files including the inventory file, the product dictionary (which translates product codes to product descriptions), the effort key file (which maintains statistics including product prices for each individual promotion generated for magazines, books, and records), and the postal guide (which maintains the correspondence between city/state and zip code*). The combined data base resides, in compacted form, on twenty-five 3330-II disks, and is backed up weekly onto tape.

This paper describes processing in BRGE, with particular emphasis on the control of information flow within the online portion of the system, a transaction journaling/backup scheme which has proved most successful in maintaining file integrity and processing continuity, and a database organization and access method (CLOD/CLAM) that permits smooth and transparent file growth and maintenance and which gives, in a 200-terminal, 150,000 transaction-per-day online system, a response time of approximately one second.

* It is an interesting historical note that zip codes were developed by the United States Post Office as an outgrowth of Time's routing codes.

Significant features, in brief, include:

- Consistent processing of all transactions regardless of source.
- Maintenance of a conglomerate transaction journal that permits rapid and transparent restart/recovery procedures.
- A multiple queue online transaction processing scheme which directs transactions to the terminal best able to handle them, and redirects transactions automatically in case of problems.
- The Cylinder Lists of Data (CLOD) file organization method and Cylinder Logic Access Method (CLAM) which permit both direct and sequential file access simultaneously, and which manage very large quantities of overflow information without noticeable loss of response speed.

TRANSACTION PROCESSING AND JOURNALING

All transactions in BRGE, regardless of source, are processed in a consistent manner. The primary mechanisms that permit this are a Front End that transforms an incoming transaction into a consistent internal format for further processing, and a Back End that performs all physical file maintenance and maintains the Transaction Journal.

Remaining processing of the transaction is performed by programs running under CICS, which make reference to the transaction via a consistently formatted 1K-byte internal transaction area (ITA). Each transaction type interacts with the Front End to place information from the transaction into predefined locations within the ITA. No program except the Front End modifies the ITA, and program problems elsewhere cannot affect transaction integrity. Once information is in the ITA, processing proceeds based exclusively on the transaction type itself, and the information it supplies. The origin of a transaction, whether from a CRT, from an optical scanner, or from the backup Transaction Journal, becomes essentially transparent at this point, except that the source is noted for error notification and later operating summaries.

As control passes among the programs that process a transaction, additional information supplementing the ITA is gathered on validation, invoicing, and statistics generated from processing. Records relevant to the processing of the transaction are obtained and put in associated buffer areas from which inquiry information is extracted, and into which requested changes are placed.

After activity on a transaction is complete, whether by satisfying the transaction through normal processing, or through error termination, control passes to the Back End.

The Back End has two primary functions. If a book/record transaction completes normally, the Back End rewrites all modified records as appropriate. In case of error termination, the Back End consults the error code posted by the program which discovered the problem and the transaction source indication, to determine where the transaction should be referred for further handling. This process is discussed in more detail later.

Regardless of errors, the Back End releases any record buffers reserved for the transaction, and writes a Transaction Journal Entry (TJE) to tape. The TJE contains an image of the entire ITA, plus information on the disposition of the transaction, and statistical and financial information for later report generation. When completion is normal, invoice data, and the text of form correspondence with appropriate blanks and options filled in may also be included in the TJE. Bills and correspondence are generated directly from the Transaction Journal, respectively daily and weekly.

Two copies of the Transaction Journal are written simultaneously on two separate tape units. Each TJE is a variable length record, up to 6250 bytes in length, and a fixed number of TJE's is written to any particular tape reel to promote interchangeability of volumes between the two copies written.

The Transaction Journal serves two purposes. The first is to maintain an audit trail for magazine, book, and record processing, and to provide a medium from which operating statistics, operator performance measures, and financial summaries can be created. The second is to provide coordinated backup of the activity of the system. In case of operating system crash, or disastrous data loss through hardware or human failure, the Transaction Journal provides a mechanism for restoring the current state of the system without needing to re-enter any transactions manually.

For soft crashes without data loss, the last reel of the Transaction Journal is scanned to determine which transaction was the last accepted from each terminal, to notify the operator where to resume. Tapes from the optical scanner are similarly and automatically repositioned. Once notification to the terminals and scanner tape has been given, data entry resumes.

In the rare case that the files are lost, the files are initially re-created from tape copies that are generated weekly. Concurrent with data entry, which resumes following initial re-creation, the Transaction Journal is mounted and read as though it were an additional transaction source. TJE's corresponding to invalid updates and to inquiries that did not require a data base change are bypassed. Valid update transactions are reprocessed, with the ITA from the Transaction Journal moved back into memory for processing. A new version of the Transaction Journal is created, reflecting both the file restoration process and the concurrent entry of new data from the CRT's and scanner.

Although plans are in progress to put magazine file updates online like book and record processing, the magazine portion of the system currently performs updates in a weekly sequential run. The system validates transactions against the data base online, and saves them in the Transaction Journal for later application. The Transaction Journal processing facility provides the ability to draw selected types of transactions from the journal (in this case, magazine updates), so that no special segregation of transactions is required.

TRANSACTION QUEUES AND ERROR PATHS

It is expected in the course of routine processing that a variety of error conditions will arise, due to misentering of information, inconsistencies between the transaction and the data base (e.g., customer sends money, file shows nothing owing), or program and system errors. In addition, conditions may arise that are not within the purview of the CRT operator who originated a transaction.

The basic philosophy of BRGE is to make these exceptions as transparent as possible to the operator if the operator cannot be expected to handle them himself. An association is made between varieties of exceptions and a series of special handling queues. When an exception arises, control passes to the Back End for further processing. If the Back End determines that the error can be handled by the originating operator (for example, a miskeyed data item, or incomplete information entered), an appropriate screen is displayed to the operator requesting the correction or clarification. Once the requested information is entered, the transaction is reprocessed.

However, if the error discovered is identified as not being handleable by the originating operator (usually a file anomaly or inconsistency requiring discretionary action at a level above that of the operator), or if any inconsistency is detected on transactions from the optical scanner, the transaction and associated information is queued for action at an exception terminal. Alternatively, an operator may force a transaction into an exception queue when, regardless of program decision, the operator does not feel capable of handling a problem transaction. Each exception terminal is staffed by personnel specially trained to manage a particular class of problem (e.g., city/zip code inconsistencies).

When a transaction is switched by program decision to an exception queue, it will appear to the originating operator to have processed normally. The design parameter here is that since the operator is not equipped to handle the condition that has arisen, there is no point in troubling him about it.

A special case is a situation in which, during transaction processing, a program interrupt occurs, as distinguished from a program-detected anomaly or decision point. The most frequent cause of such an interrupt is simply a bug in one of the transaction application programs. In this case, the transaction and associated information is placed in a special queue for programmer attention. This convention has proved enormously useful for helping to detect and eradicate bugs that are highly data and circumstance dependent. Having available the exact data that generated an interrupt makes this level of debugging far easier to deal with than having only the knowledge that an interrupt occurred. And again, the originating operator is not distracted or delayed by a condition over which he has no control.

At the terminals that draw from the exception queues, whether they are exception terminals or programming terminals, a consistent mode of operation is maintained. When a terminal assigned to a particular exception queue indicates its readiness to proceed, the interrupted transactions are called up and re-executed one at a time.

In most cases, the error condition that initiated the transaction's being placed in the queue will recur. Now, however, the terminal initiating the transaction is the terminal capable of handling the problem. An appropriate error report is given, and the exception operator can take the action he sees fit including, when necessary, re-directing the transaction to another queue. In addition, it will sometimes happen after one problem has been dealt with that another problem will arise, requiring the requeuing of the transaction to handle the subsequent problem.

In some cases, when a queued transaction is called up for re-execution, the error will not recur. Often the cause of this non-replication is that an anomaly between transaction and file has disappeared through other file maintenance that took place between the original exception and the re-execution.

A common program interrupt is one due to a deadlock situation that was detected by BRGE. In its simplest form, deadlock will occur when transaction X already has record A and also requires record B in order to proceed; and transaction Y has record B and requires record A to proceed. If a set of transactions are in this kind of contention, BRGE cancels all but one of them, and requeues the cancelled transactions for later processing. Cancelling the transactions releases the records under their control to permit the remaining transaction to complete. When called back up, a cancelled transaction will typically not encounter record contention again, and will simply run normally to completion.

Personnel at the exception and programming terminals will be unaware of transactions that upon re-execution complete normally. Their attention will be requested only when the error recurs.

DATA BASE

The BRGE data base consists of the customer/subscriber file and a series of ancillary files.

- *Customer File.* This file consists of five billion characters of customer information, one record per customer, in segmented form. Magazine data is currently kept separate from book/record data. A customer will appear in the file for each of his magazine subscriptions, and once if he is a subscriber to one or more book/record series. Each record has a root segment containing identifying information, credit status, and a summary of information in the remaining segments (if any). Magazine records have only a root segment. In the book/record portion of the file, a segment represents purchases in a book or record series. Plans are in progress to reorganize magazine information into a unified scheme like books/records.

Book/record customer records are processed using a segment logic facility. In memory, each segment appears as an individual variable length record, locatable by application programs in well-defined spots in memory. The segments are gathered together and combined into a single variable length record for peripheral storage. When read back into memory, the combined record is redistributed into individual segments for processing, but only when a segment other than the root is required.

- *Inventory File.* This file applies primarily to the book/record portion of BRGE, and contains information on the quantity and location of each product available for sale. Time currently maintains an inventory of 646 different titles in 13 warehouses in the U.S. and Canada, shipping approximately 1.25 million items a month. Up to three hundred different inventory items are included in the inventory file for each warehouse location. Online order entry makes direct adjustments to the inventory file.
- *Product Dictionary.* The product dictionary maintains, for each book and record series, a translation between the series code and descriptive names for the series. Several descriptions of varying lengths are maintained, for use in a variety of contexts. Similar code-to-description tables are kept for each item within a series.
- *Postal Guide.* The postal guide falls into two parts. The first is an alphabetical listing of U.S. cities, states, and zip codes, with variants and probable misspellings. A corresponding Canadian list is also maintained. When a city/state/zip is entered into the system it is verified through this alpha listing. If no match is found, the entering transaction is rerouted to a postal specialist for examination and correction. A special keying feature is used in alpha lookups: The access method provides for access on partial keys; that is, presentation in sequence of all records whose leading key positions match a given partial key. But in addition, records in a file may have designated short keys: Any requested key whose leading portion matches the designated short key will select the record containing the key. In particular, if a unique city name or a leading portion of such a name corresponds to exactly one zip code, it may be designated as a short record key. Regardless of misspelling or variation of the state, if the city is entered correctly with zip code, a postal guide match will be made. This feature is also useful for cities with multiple zip codes. For example, Chicago can be entered (along with variants) as *Chicago, IL 606* and an entry with the proper leading digits of the zip code present would cause a hit on the file. As added verification in this case, a range check is also made on the trailing digits, once a file hit has been accomplished, eliminating the need to enter all 80 Chicago zip codes separately into the postal guide.

  The second portion of the postal guide is a numeric zip-to-city/state translator. Once data entry is verified, the zip code is maintained in a customer record, and city/state information is excluded. City/state is reassociated with a record (for label printing purposes, for example) through this second postal guide section.
- *Effort Key File.* An effort key is a code assigned to each individual promotion made in magazine, book, and record marketing efforts. The effort key file maintains statistics on the results of each promotion to date, and includes the specification of rules for fulfilling the promotion, including prices for the items promoted, sequence for shipping of items in a series, payments terms, special conditions, etc. Terminals in Time's New York corporate offices monitor the daily activity of this file to help guide the overall marketing effort. Online order entry causes immediate update of effort key statistics, so that this file is always current to the moment.

FILE ORGANIZATION AND ACCESS

All of the files described in the previous section are maintained together as a single OS file to minimize job control, system control information, and interaction with the operating system. The logical independence and identity of the files is maintained via a file organization method called Cylinder Lists of Data (CLOD), and a corresponding Cylinder Logic Access Method (CLAM). CLAM is an EXCP-level access method that operates on IBM 360's and 370's under OS and VS, to permit sequential and random file access.

The only consistency that CLOD requires among the files it controls is that they use a common block size, and are located physically on the same variety of storage device (3330-II drives at Time). CLOD maintains, for the disk extents made available to it, a list of available cylinders for allocation to the logical files it controls. Similar lists are developed for each file as cylinders are allocated to the individual files. Records within the files are maintained in order by key. Ignoring overflow for the moment, reading records cylinder-wise according to a file's cylinder list results in the file's records being read in order. Any file may be maintained either in ascending or descending key order. The cylinder list itself is in correspondence with the allocation sequence, but has no intrinsic numeric order.

When a file requires an additional cylinder, it is allocated from the extent that has the fewest cylinders currently allocated. One of the effects of this allocation scheme is that files tend to be evenly spread over all available extents, minimizing the contention on a particular volume for access to a heavily used file. Additional volumes and extents may be added without interrupting processing, and CLAM will display operating and access statistics on request, to assist in performing load leveling. CLAM provides the ability during normal execution to transfer cylinders of data among extents to spread out heavily accessed portions of a file.

The key file consists simply of the sequence of first keys from each block. Given the cylinder list and the block size, the correspondence between the nth key in a key file and its block is readily and speedily established. Key files may be kept on a direct access medium, or in memory. In BRGE, which uses full-track blocking, only the customer/subscriber file's key file is kept on peripheral storage.

One additional level of indexing is always present, in the form of a core index to the key file. The core index is an index to the blocks of the key file, exactly in the same manner that the key file indexes the main file. With the choice of a reasonable block size, there is never any reason to keep this core index anywhere but in main memory. For the subscriber file, with five billion characters of data across twenty-five packs, the core index occupies only about 3K bytes of storage.

CLAM resembles ISAM and VSAM in that it permits both sequential and random file access. However, CLAM operation, particularly regarding the handling of overflow, is quite different. The handling of overflow records is designed to minimize the number of additional seeks needed to find the records. This is accomplished by keeping records that overflowed from the same block physically together to the greatest degree possible.

To describe overflow processing, two levels of record collections in addition to the block and the logical record are needed: the *bundle* and the *span*.

A bundle is simply a related collection of one or more logical records. Several bundles may be combined to form a block.

A span is the collection of all logical records addressed by a single entry in the key file. If a main data block has no associated overflow records, a block and a span are identical. However, if overflow records are associated with a block, the span includes the main data block plus all associated overflow records. The records which constitute a span are logically all in order by key.

At the point that a main data block overflows, a bundle is created from the logical records that will not fit into the main data block. In particular, trailing records from the end of the span are placed into an overflow bundle, leaving a full block of records from the beginning of the span in the main data area. This overflow bundle may be combined with other overflow bundles to form a block within an overflow cylinder. (All files controlled under the CLOD share the same overflow cylinders). A flag and pointer are included with the main data block to direct CLAM to the appropriate overflow block when necessary. Such an additional access will be made when the flag is set, and when (assuming ascending file order) a desired record's key lies between the key of the last record actually in the main data block and the key beginning the next sequential span.

CLAM's speed in locating overflow records comes largely from a very dynamic control of the overflow area, insuring that overflow records from the same span stay together. When adding a new record to an overflow bundle, it may happen that this newly enlarged bundle, plus the other bundles from the same overflow block no longer will fit into a single block. In this case, the just-modified overflow bundle is split away from the other bundles, and placed into a new block which will accommodate it.

As a consequence, finding an overflow record never requires more than one additional seek, unless more overflow records exist for a single span than will fit into an entire block. In addition, if space becomes available in the main data block through deletion of a main block record, the span is reapportioned to move overflow records back into the main data block. Freed space is immediately available for reuse.

Although CLAM/CLOD include provision for introducing slack space into main data blocks at the time that a file is created or reorganized, the speed of handling overflow generally makes such provision unnecessary. Situations have been observed in which the presence of more than 2,000 cylinders of overflow records had no noticeable effect on online response time.

All files organized together under CLOD share the same pool of buffers (hence the requirement for consistent block size). A program may specify whether CLAM should return the address of a record within a buffer for direct processing, or move the data to a program-defined area. When a data buffer is freed, it is linked to the end of the chain of available buffers. If a request is made for a record in a freed buffer in which no errors appear, before the buffer is reused, the request will be satisfied without any I/O operations, by reactivating the buffer and removing it from the available chain.

CLAM permits a file to be accessed simultaneously as though it were two independent files. It is possible for a program to process a file sequentially, and also periodically request a record from the same file randomly by key, based for example on a flag and pointer present within the record being sequentially processed. Sequential processing, when resumed will continue with the next logical record.

CLAM has its own facilities for providing exclusive control of blocks to programs performing update activities, and includes tests for determining deadlock situations. A program specifying read-only file use is not restricted from access to a file, even when an update is taking place. A specific block being updated is, of course, locked up for the duration of the update.

Other facilities contained directly within the access method include:

- Record compaction/expansion exits. Routines may be supplied for any CLOD logical file to perform compaction and expansion of logical records. On input, CLAM will locate a compacted record, call upon the expansion routine supplied for this file, and return control to the program requesting the record as though an uncompacted record had been read from the file directly. The converse operations are performed for output.
- Record segmentation facilities. CLAM will accept a collection of record segments, each with the appearance of an individual variable length record, and combine the segments into a single variable length record for placement in the file. The record may be composed of up to 64 segments corresponding to a bit table or to a combined key area at the front of a combined record. CLAM will resegment such a record for program use upon request. Both segmentation and compaction may apply to the same file.
- Handling queue files. CLAM commands provide for the special handling of queue files. Basic facilities include the ability to add records (messages, transactions) to the end of the file, and to draw records from the front of the file in standard first-in, first-out fashion. Multiple sources and draws may be operating on a queue simultaneously. In addition, facilities are available to examine records beginning at the front of the queue, or to search for records with special keying within the queue. Records are actually deleted from the queue only on specific command to do so.
- Access from all languages. Primary use of CLOD and CLAM to date has been via assembly language, in which a complete set of macros are available to direct file activity. In addition, a subroutine SEICAM, accessible via standard call from all high level languages provides these languages with EXCP-level CLAM access to CLOD files. Commands are included to read and write records, open and close logical files, add extents, perform segmentation and combination, and to retrieve statistics on file activity.

HISTORY AND CONVERSION

Until 1974, Time operations were carried on via emulation of IBM 7070 on an IBM 370/158. Each magazine and book/record series had its own masterfile, and each was processed separately in a classical card-tape environment. Time developed a design for conversion to a unified system, but found that operating costs and times were unacceptable using standard vendor-supplied software. Implementation of the unified design through an extended CICS system with new file organization and access techniques was accomplished through a combined effort of Time, Inc. personnel and SEI Computer Services. Principal design for CLOD/CLAM was done by T. R. Shields of SEI. Implementation began in May 1973, and conversion and production on the new system both began in April 1974. Conversion was accomplished by a straightforward translation of old masterfiles into BRGE transactions. BRGE itself was used to process the transactions and thereby enter the content of the old masterfiles into the unified customer database. Product files were so converted over a period of about five months. At the point that a product was designated for conversion, activity on the product was suspended while conversion took place, after which activity resumed under the new system. Conversion suspensions of activity were typically two to five days, but only on the one product being converted. For all other products, activity continued as usual. The system was fully operational and converted in September, 1974. At that time BRGE ran on an IBM 370/158 under SVS. The system is currently running on an IBM 370/168 MP under MVS.

One additional point on the conversion process is worthy of note, namely that conversion using the new system itself provided a thorough check-out of system features and intercommunications. Particular use was made of the programming exception terminals for locating and fixing bugs. Although considerable quantities of transactions would appear in the program bug queues, the catching and fixing of a bug would typically cause the uninterrupted flush-through of a large percentage of the queued transactions. During the conversion process, a peak transaction processing rate of 80,000 transactions per hour was observed. The system retains the capacity to process at this rate, but normal operations have not to date been able to generate transactions at a rate corresponding to the system's full capacity.

BIBLIOGRAPHY

1. IBM Corporation, *IBM System/370 Principles of Operation*, Form GA22-7000
2. IBM Corporation, *Customer Information Control System* (CICS/VS), General Information Manual, Form GH20-1280. System/Application Design Guide, Form SH20-9002
3. SEI Computer Services, *SEICLOD Data Management System*, User's Guide, October, 1975.

Generalized software for translating data*

by EDWARD W. BIRSS and JAMES P. FRY
University of Michigan
Ann Arbor, Michigan

ABSTRACT

Many data processing installations are confronted with the problem of data conversion. Some of the conversion problems are conversion of files foreign to the installation, conversion of files into a data base management system format, and conversion of all data to upgrade hardware or software. Simple file organizations pose few conversion problems, while logically and physically complex data bases emphasize many conversion problems. The current approach of writing specific translation programs is time consuming and frequently inaccurate; a new approach is desirable.

To address these conversion problems, The University of Michigan Data Translation Project has developed a generalized translation methodology. This methodology has been applied in the development of several prototype data translators. These translators have progressively advanced the physical transformation capabilities (reformatting) and the logical transformation capabilities (restructuring). The reformatting capabilities of the translators include the ability to access and modify the physical storage structures which support sequential, indexed sequential, and network organizations. The restructuring capabilities allow complex restructuring of lists, trees, and networks.

Future extensions to the translation methodology include the decomposition of the translation process into small, but specific steps. Languages would be developed to address each of these small translations, and could lead to a generalized accessing mechanism and a data interchange form.

INTRODUCTION

The computer field is a rapidly expanding area with advancements in computer hardware and software technology that are paced by the increasing sophistication and awareness of the users. Expanding utilization of computer facilities and the ever increasing application demands continue to usurp valuable resources. Caught in the middle of this situation is the data processing manager, who must satisfy the demands of the user community and yet maintain the economic attractiveness of the computer system.

One of the many problems facing a data processing installation is the conversion and transformation of data bases. Typically this problem ranges from the conversion of computerized files from other installations (foreign files), the restructuring and reformatting of extant data bases, to the translating of data into various forms required by different applications.

Concomitant with the increased use of data base management systems, the data base administrator is faced with the necessity of creating and/or integrating existing files into data bases or of restructuring existing data bases. The latter capacity, while necessary to make effective use of the data base, is not typically found in data base management systems.

To take advantage of the economic benefits of new hardware/software capabilities and data base management systems, the data processing manager and the data base administrator need a variety of data base conversion tools. The current manual approach of writing specific programs for each conversion is both time consuming and costly. A fresh approach to the problem is therefore needed.

In order to address the data conversion problems of the Data Processing installation, a new software technology has been developing over the past five years called data translation. One group developing a data translation methodology is The University of Michigan Data Translation Project, where several prototype data translators have been developed. These developments and related activities are aimed at providing both the data processing manager and the data base administrator with facilities for foreign file conversion, data base integration, data base restructuring, and data conversion resulting from upgrading hardware or software and changing user requirements.

The purpose of this paper is to describe the progress at The University of Michigan on data conversion based on the development of a data translation methodology. The paper identifies those areas of data base

* This work was supported by the Department of Defense, Defense Communications Agency, WWMCCS ADP Directorate, Reston, VA Contract No. DCA 100-75-C-0064.

conversion in which the translation methodology has been successfully developed and is ready to be applied to current data processing problems. It describes current areas under development and those which are in need of further research.

The paper begins by describing the current research on data base conversion and develops the data translation approach. Next, the evolution of data translators at The University of Michigan is traced in terms of their logical and physical capabilities and in terms of generality achieved. The paper concludes with some observations on a generalized data translation methodology and enumerates those areas which need to be researched.

BACKGROUND AND APPROACH

Within the last five years, a means of attacking the data base conversion problem in a general manner has been proposed.[1–4] It is interesting to observe that all of these efforts employ some degree of generality and are based on a descriptive approach, which describes the source and target data bases and the necessary transformations required to derive the target data base instances from the source. These descriptions, couched in a high-level declarative language, provide the basis for two implementation approaches for a generalized translator. The *interpretive* approach develops a generalized processing program,[1] and the *generative* approach creates a specific translation program[4] for each conversion.

The University of Michigan Data Translation Project's approach,[5] emblematic of others, consists of two steps:

1. The user specification of the necessary descriptions, and
2. The execution of data translator based on these descriptions (Figure 1).

The user supplies descriptions of the logical, physical, and relational aspects of the source and target data bases, along with the specifications of the restructuring transformations required to map source data into the target data. Two languages were developed to provide these descriptions; the Stored-Data Definition (SDDL) which is used to describe the source and target data bases, and the Translation Definition Language (TDL) used to describe the restructuring transformations.

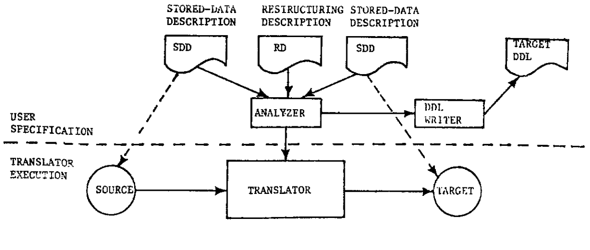


Figure 1—Data description approach

The SDDL is based on the language proposed by Taylor.[6] This high-level language uses a modified CODASYL model of data. At first glance, the SDDL is similar to the data description language of a data base management system, but a closer look reveals several important differences. The stored-data definition language, based on common data definition practices, is actually an extension of the logical DDL to the more physical implementation aspects. It describes the mapping of the user logical structure to physical storage structure, the mapping of the physical storage structure to a storage device, and the access paths to the data.

The Translation Definition Language, on the other hand, deals primarily with logical transformations of data and describes the translation of source instances to target instances. The language was developed at The University of Michigan and began as a simple association list of source item names and target item names, but it has since developed into a powerful restructuring language. Detailed discussions of these languages can be found in References 5 and 7.

The SDDL and TDL descriptions are processed by an Analyzer which produces an Encoded Stored-Data Description (ESDD) and an Encoded Restructuring Description (ERD) respectively, which in turn are used to drive the translator.

An anxiliary module which need not be generalized, the DDL Writer, uses the Encoded Stored Data Definition of the target to construct a data definition of the target data base in the language of the target DBMS. Major benefits of this module include not only the immediate use of the data base by application programs, but more importantly, the user verification of the target description. The DDL Writer allows the user to verify in a language familiar to the user that the target data description is consistent with his view of the target data base.

The second step in the translation process is the physical transformation of the source data into the target data. Driven by the data descriptions prepared in the first step, the second step employs three components; a Reader, a Restructurer, and a Writer (Figure 2). The Reader accesses the source data base, the Restructurer transforms the source data into a form suitable for the target, and the Writer creates the target data base.

The Reader module, driven in part by the source ESDD, performs many functions; sequentially accessing physical records, physically deblocking these records, logically identifying their components, and automatically creating an internal data base processable by the Restructurer. In dealing with complex data base structures, the Reader must keep track of

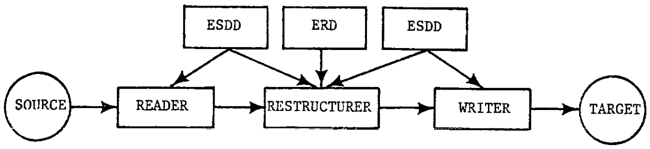


Figure 2—Components in the translation process

access paths between these unblocked records so that the Restructurer is provided with an accurate representation of the source data base.

The Restructurer accesses the internal representation of the source data base and converts it to a representation of the target data base. The conversion from source to target is a transformation of the logical structure and is directed by the ERD which contains the restructuring specifications.

The Writer, driven in part by the target ESDD, creates the target data base by constructing target data derived from the Restructurer's internal data base.

The approach by The University of Michigan, described above, is an interpretive translation approach in which stored-data descriptions provide parameters to a generalized translation algorithm. An alternative approach, a generative one, creates specific translators for each translation[4] (Figure 3). There is, of course, a hybrid approach in which some of the components in the model of the interpretive approach produce executable code, while other components remain interpretive.[2] The interpretive approach was chosen because it provides a good research tool and facilitates the building of series of data translators.

EVOLUTION OF DATA TRANSLATORS

The translator development at The University of Michigan has designed four translators: a Prototype, Version I, Version II, and Version IIA. Only three of these designs, however, have been implemented; Prototype, Version I, and Version IIA. The various designs were developed to address different problem areas, and each translator design has produced a major contribution to the generalized translation process.

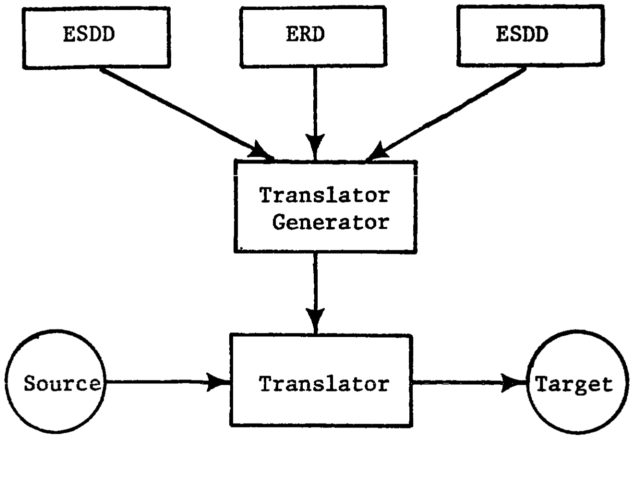


Figure 3—Alternative DDL driven data translation

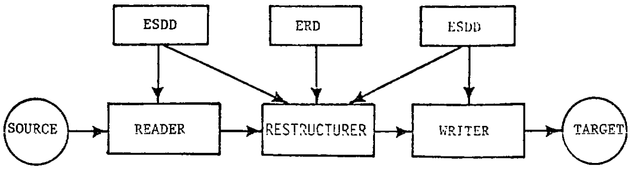


Figure 4—Prototype translator

The translator's capabilities are divided into two categories; physical and logical. The Translator's physical capabilities are characterized by the Input/Output processing ability of the Reader and Writer. The translator's logical capabilities are measured in terms of the restructuring proficiency of the translator's Restructurer component.

In the remaining portion of the paper, the purpose, architecture, major contribution, physical capabilities, logical capabilities, generality, and implications of each design are discussed.

Prototype translator

The prototype translator was developed to provide a framework for further research and to verify that the proposed technical approach was sound.[8] The architecture of the prototype translator was very similar to the translator model described in the previous section (Figure 4). The Restructurer, however, only performed an identity transformation, merely associating source item names with target item names.

Prototype's physical capabilities

There was not a great deal of generality in the prototype; the Reader module only accessed one source data base; and the Writer module only created a single output. The source data base was a NIPS data base unloaded in a variable blocked sequential format (Figure 5A). NIPS ran on the IBM 360/370 series of com-

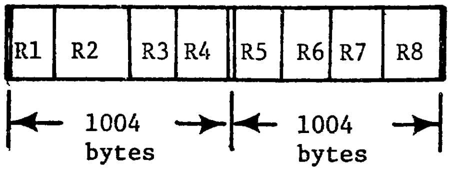


Figure 5A—NIPS format

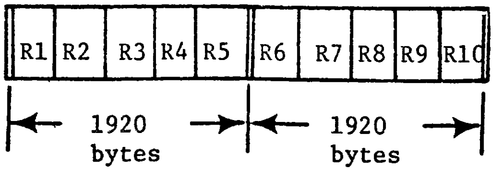


Figure 5B—H6000 system standard format

puters and allowed two-level hierarchical structures with a maximum 256 different groups.

The Writer was also straightforward. It constructed a structure similar to the NIPS data base structure, in the Honeywell 6000 System Standard Format. The H6000 System Standard format was similar to the IBM variable blocked format (Figure 5B).

The Reader and Writer operated with sequential media, and physical concatenation (i.e., a simple mapping algorithm) was used to maintain the hierarchical relationships in both systems. Both systems used similar mapping, so reordering of record types was not necessary.

Prototype's logical capabilities

The Prototype translator does not perform restructuring, and when no records were eliminated, a one-to-one correspondence existed between group instances in the source data base and those in the target data base (Figure 6). Consequently, the Restructurer only processed one group instance at a time. It performed item conversions as necessary and allowed the user to modify the order or storage representation of items within a group.

Generality

The only general module (not coded specifically for the NIPS to H6000 System Standard Format) was the Restructurer. Its generality was possible because of the level of insulation provided by the encoded forms of the SDDL and TDL languages which provided all the necessary information for the table driven architecture of the Restructurer. In contrast to the Restructurer, the Reader and Writer were very specific and used the operating system facilities to perform input/output.

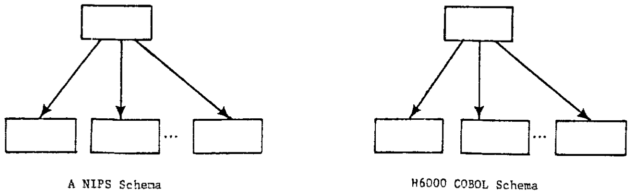


Figure 6

Design implications

Although the prototype performed a very simple translation, its major contribution was the framework for a series of generalized translator implementations. The Prototype design provided a consistent set of interfaces: between the languages and translator modules, through the encoded tables, and among the translator modules by a set of table access routines and common data formats. Throughout the development of generalized translators, this basic architecture has proven to be technically sound.

Version I translator

The Version I translator (Figure 7), although very similar to its predecessor, made some progress toward generality. Version I was developed to perform both the forward and reverse translations (NIPS→System Standard format and System Standard format→NIPS). Furthermore, the COBOL System Standard format files were constrained to a format acceptable to WWDMS, a data management system on the H-6000 computer system.

The purpose of this translator was to further verify that the translation methodology was sound and to demonstrate that the translation was reversible without loss of information. The latter proved to be an interesting technique to verify that a translation was correctly performed.

Version I's physical capabilities

The Reader and Writer were required to access and create both NIPS and H-6000 formats (Figure 5B). The Reader and Writer modules were parameterized to enable them to read and write both organizations. Additionally, the Reader was generalized by taking greater advantage of the Encoded SDD. Thus, the Reader's capabilities were generalized to handle most sequential tape formats.

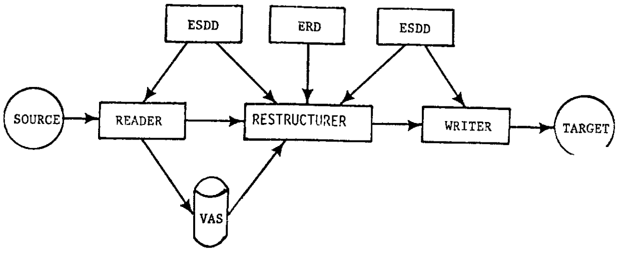


Figure 7—Version I translator

Version I's logical capabilities

The major difference between the Version I translator and the Prototype translator is the addition of a basic restructuring capability. The major objective of Version I was to demonstrate that the inverse translation (source-target-source) was feasible and indeed produced the "original" data base. Restructuring was often required, because the source, COBOL-WWDMS data bases, had eight levels of hierarchy, but the target, NIPS, could only handle a two-level hierarchy. To permit such translation, the first restructuring capability developed was *compression* of hierarchical levels. For example, a two-level target schema was constructed (Figure 8) from the multiple level source schema. In order to preserve the information of the source data base, the Restructurer created a new group PROD-PARTS and replicated the PARTS information in this new group of target data base.

Generality

The increase in generality of the Version I translator occurred primarily in the Reader and Restructuring components. A new section of the SDDL was developed to provide a better description of physically sequential media. This additional section permitted the deblocking, spanning, and identification operations to be generalized. The Restructurer increased in complexity through addition of the parsing and restructuring functions. These additions to the Restructurer were driven by additions to the SDDL and TDL. Thus, these language additions maintained the generality of the Restructurer component.

The Restructurer became complex, but the software architecture insulated it from the source and target data bases. Its complexity was isolated and could be addressed by revising single modules, as opposed to developing a new translator. The compressing restructuring capacity proved to be quite general. Not only could adjacent levels of the hierarchy be combined, but multiple levels of the hierarchy could be combined into a single level.

Design implications

The Version I translator design required the capability to compress schemas. The compression operation required access to an entry instance. An entry instance can be very large, and a mechanism was added to the translator to give the Restructurer direct access to the entry instance. This mechanism was termed a Virtual Address Space (VAS). The Reader constructed the VAS from group instances contained within one entry instance.

Version II translator design

The objective of the Version II design was to increase the input and output data base classes and to expand the restructuring capabilities of the translator. The architecture of the Version II design was different from the previous translators (Figure 9).

The main difference between the Version II translator and previous designs resulted from the realization[1] that the more complex restructuring operations became, the greater the volume of data required. Such complex restructuring also required broader accessibility to the data base. Because the Restructurer requires many access paths to data, some of which are not available in the source data bases, an internal form of the source data was designed, the Restructurer Internal Form (RIF). A data management function was also incorporated to manage the RIF which allowed the Restructurer to access the RIF directly.

Version II's physical capabilities

The capabilities of the Reader and Writer were to be increased to handle sequential, indexed sequential (ISP), and network (IDS) structures (Figure 10 A,B,C).

These capabilities were to be implemented using a general control structure in the Reader for invoking the specific accessors for the different organizations. The lowest level components were parameterized.

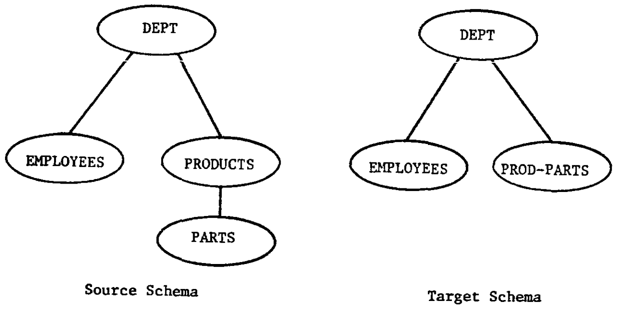


Figure 8

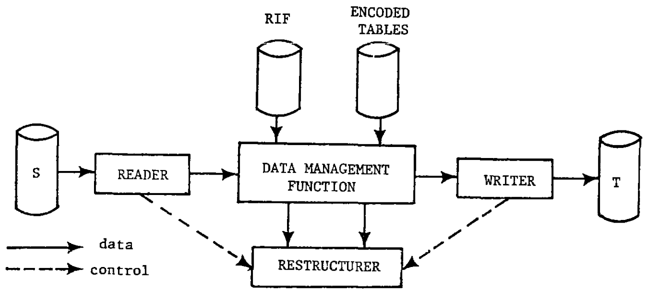
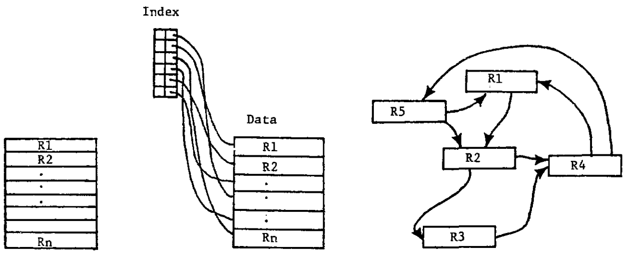


Figure 9—Version II design



SEQUENTIAL Figure 10A

INDEXED SEQUENTIAL Figure 10B

NETWORK Figure 10C

Figure 10

Version II's logical capabilities

The general problem in restructuring data bases was a very difficult and complex one with no established solutions. The initial research[9] indicated that restructuring for hierarchical structures was feasible, but was yet to be implemented. Consequently, Version II was designed to perform restructuring within hierarchical structures, which could possibly be subsets of network structures.

The Restructurer's design facilitated the creation of any target logical structure derivable through repeated applications of Expansion, Compression, Partitioning, and Merging restructuring operations on the source logical structure. Examples of these restructuring operations are shown in Figure 11.

Generality

Extending the capabilities of the Reader and Writer from Version I's sequential representations to the sequential, indexed sequential, and network representations of Version II had an impact on generality. The sequential Reader of Version I was reasonably general, but the implementation of the SDDL's extended capabilities to handle the representation of complex logical structures on disk media proved extremely difficult. Consequently, accessors, low level routines coded specifically for each organization, were designed for the more complex input/output operations and interfaced with the general software in the Reader. The Restructurer provided an increased level of generality by providing a comprehensive set of restructuring capabilities for hierarchical structures.

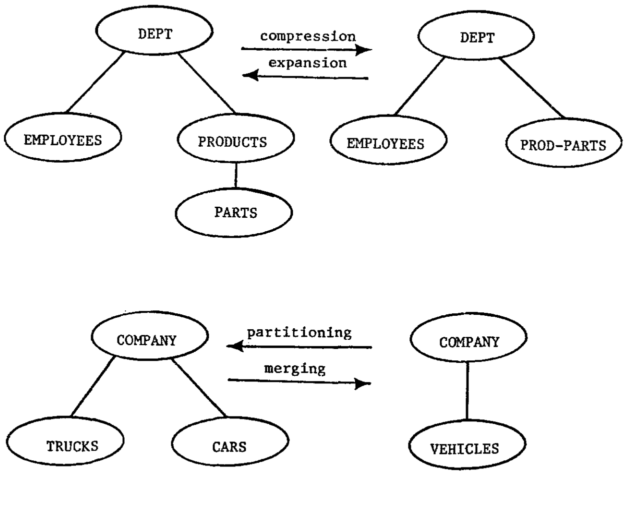


Figure 11—Restructuring capabilities

Design implications

The complex restructuring operations desired for the Version II frequently required access to the entire data base. This requirement led to the development of a Restructurer Internal Form (RIF) in the Translator and the result that the read function had to be performed completely before the Restructurer was envoked. In order to help manage the complex and voluminous RIF data base, a data base management system[10] was added as a major component of the Translator's design. This DBMS was used not only to provide direct access to information stored in the RIF, but also was used to manage the internal tables.

Version IIA translator

After evaluating our design of the Version II translator and some actual data base reorganizations, a decision was made to emphasize restructuring at the expense of the physical capabilities—reading and writing. An alternative design, the Version IIA translator, was developed which focused on the restructuring within the I-D-S data base organizations.[11] The specific nature of the translator allowed the design effort to focus on user orientation as well as restructuring. It was decided that an expanded version of the I-D-S DDL could be used for the description of the source and target data bases. The I-D-S DDL was augmented by using Level 66 descriptions. Another DDL analyzer, specific to I-D-S, had to be developed, which not only processed the extended DDL but also produced the Encoded SDD (Figure 12).

Version IIA's physical capabilities

Since the Version IIA had to restructure I-D-S data bases, the input/output class of the Reader and Writer were limited to these structures (Figure 10C). However, the Reader was generalized to populate the RIF from multiple source I-D-S data bases which allowed the integration of I-D-S data bases, the addition of new data and structure, and the addition of new relationships among existing data instances.

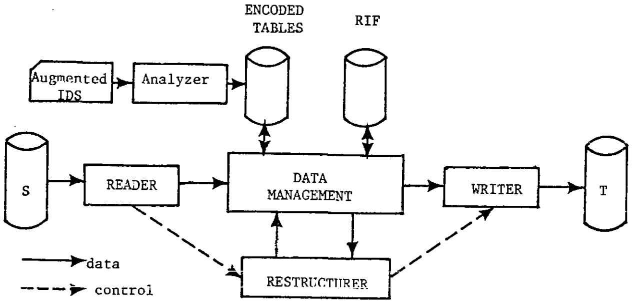


Figure 12—Version IIA translator

The Writer, since it was coded for I-D-S data bases and used the I-D-S access methods, could perform limited storage optimization through reformatting of the target data base. The various storage parameters (place near, . . .) available in the I-D-S DDL could be used to create more efficient target data bases.

Version IIA's logical capabilities

The Restructurer made a major research and development step by extending its capabilities to network structures. The Restructurer implemented was extremely powerful and could not only change the implementation structure of existing I-D-S relationships, but also had the capability to create structures more powerful than I-D-S I could process. Some of these restructuring capabilities developed in addition to the Version I capabilities are shown in Figure 13A-13C.

Generality

Overall the Version IIA translator is less general than the Version II design chiefly because the Reader and Writer were tailored to process I-D-S data base thereby simplifying the coding effort. The specific approach to the Reader and Writer stemmed from the Version IIA emphasis on the Restructurer, and the difficulty of implementing the extended SDDL to describe complex structures residing on disk media. The Restructurer, however, maintained its generality by the table driven architecture and greatly expanded its capabilities from hierarchical to network structures.

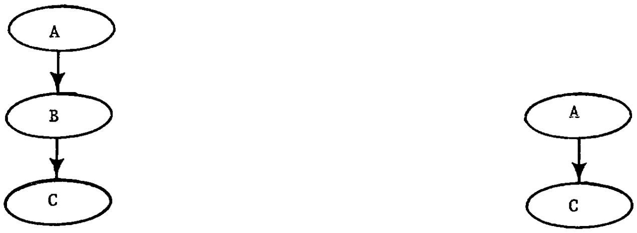


Figure 13B—Creating a relational entity AC from AB and BC

Summary of capabilities

During the evolution of the Michigan data translators, both the Physical and Logical Capabilities of the translators were increased in incremental steps. The capabilities of the Reader and Writer modules have been extended from sequential to network data base representations with some loss of generality. Restructuring capabilities have increased from simple renaming of structures, through hierarchical restructuring to network restructuring. Although it was necessary to expand the Restructurer's accessibility of the data base from a single record to the entire data base in order to achieve these sophisticated restructuring capabilities, the generality of the Restructurer was preserved through the RIF and the DBMS. In addition to the translation capabilities, it became clear that additional capabilities of adding data and new relationships to the data base were necessary and implemented in the Version IIA. The evolution of the basic capabilities of the various Michigan translators are summarized in Figure 14.

REFLECTIONS ON THE DEVELOPMENT OF DATA TRANSLATORS

Over the past four years, The University of Michigan Data Translation Project has developed a series of increasingly more general data translators based on a stored-data definition language approach. In reality, a general data translator is a series of three interpre-

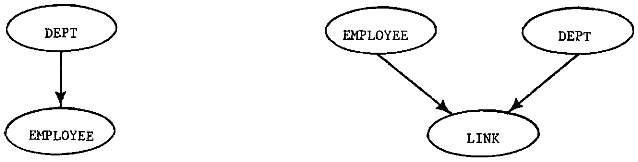


Figure 13A—Changing from duplicated data to non-redundant representation

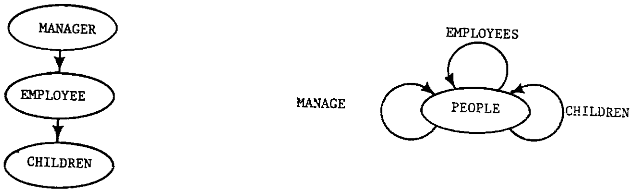


Figure 13C—Changing implementation of the relation (involving merging also)

	PHYSICAL CAPABILITIES	LOGICAL RESTRUCTURING	QUANTITY OF DATA	OTHER FEATURES
PROTOTYPE	SEQ	NONE	RECORD	
VERSION I	SEQ	COMPRESSION	ENTRY	GENERALI7FD SEQ READER
VERSION II	SEQ,ISP,IDS	HIERARCHICAL REST.	DATABASE	
VERSION IIA	IDS	NETWORK REST.	DATABASE	DATA ADDITION INTEGRATION OF DATA BASES

Figure 14—Comparison of capabilities

tive mini-translators; a Reader, Restructurer, and Writer. In order to perform a translation from a source data base to a target data base, the Reader accesses the source data base, the Restructurer performs the required logical transformations on the source and develops the target structure, and the Writer constructs the target data base. These mini-translators may be categorized according to the type of transformation performed—either physical or logical. The Reader and Writer perform primarily physical transformations, while the Restructurer performs a logical transformation. The requirement to perform complex logical transformations in the Restructurer had a substantial impact on the architecture of the Translator particularly in the areas of data handling and representation. The complexity of the transformation required direct access to all components of the data base and an internal form of data was developed for the Restructurer. To address representation of complex logical structures for the internal form and to provide direct data access, a data management facility was implemented within the Restructurer.

The efforts at The University of Michigan and Honeywell have established that it is technically feasible to convert data bases created by one DBMS into a form required by another DBMS in a reasonably general manner. The generality is achieved through a stored-data definition language which directs the translation process. The translation can be quite complex; for example, changes in word length, blocking factors, data representations, and logical structures. Additional translator capabilities include: the addition of data, addition of relationships, and integration of data bases.

The economic feasibility of the SDDL driven approach has been demonstrated for simple data base structures and representations. However, as the generality of the translation process increases, the complexity of the language increases correspondingly. An alternative approach would use the host DBMS DDL to produce the ESDD (as in the Version IIA). This would simplify the user's task but would require a DBMS specific program to construct the ESDD. Such an approach would allow translation with only one unfamiliar language, the Translation Definition Language.

FUTURE DIRECTIONS AND DEVELOPMENTS

The interdependency of the logical and physical aspects of data, coupled with the dependence of the physical transformation on the hardware/software environment, makes generalized data base conversion a complex but challenging problem. The data translation methodology developed separates this problem into a logical and physical transformation processes which allow specific modules to address these transformations. Significant results have been achieved in generalizing the logical transformation process—the research, development and implementation of a generalized Restructurer. Although a generalized Restructurer has been shown to be technically feasible, its economic justification has not been demonstrated. The economic feasibility of the approach is in doubt because of the slow execution speed of the Restructurer (Version IIA), however, an optimization effort has been initiated which should greatly improve the efficiency of this process.

The achievement of complete generality in the physical transformation process has yet to be achieved. It appears much more difficult since the deeper one goes into the actual representation, the more complex the description process and the implementation of the physical transformation modules. For example, inverted structures that are implemented in SYSTEM 2000 and ADABAS are extremely difficult to translate because the SDDL for such organizations must not only describe the data, but also describe the format and semantics of the indices. Extending this part of the SDDL, the Reader, and the Writer modules to handle these organizations would require a substantial effort. At this point in the research, the development of a completely general physical transformation modules does not appear to be economically justifiable.

Building on this result, a vehicle for further decomposing this complex process into its physical and logical components needs to be developed. One such candidate is a common Data Interchange Form (DIF) for the transferability of structured data bases. Although the development of the Data Interchange Form is not easy and requires an additional transformation in the translation process, it would nevertheless be a simplifying ingredient. The design and specification of a common Data Interchange Form would result in:

(i) Separation of the hardware/software environmental considerations;
(ii) Development of specific accessors and constructors to address the physical transformation process;
(iii) Distribution of the effort between the source and target machines;
(iv) Development of a more efficient internal form of data for the Restructurer.

Further, the development of a Data Interchange

Form would facilitate the general transportability of structured data bases. Such transportability would not only occur using interchange media (e.g., tapes), but also occur over communication networks (e.g., ARPANET).

Another argument for a Data Interchange Form is that it allows the translation process to be more explicitly decomposed. The stored-data definition language could be divided according to individual transformation, which would allow different forms and types of the language to address the differing levels of detail in the translation process. The translation modules would be more dependent on their respective environments and, consequently, easier to build. Further, such a decomposition would allow optimization to take place at different stages in the translation process.

The design and specification of a common Data Interchange Form still requires research and development. It must be self-describing in that it would access interpretively, independent of the environment in which it was created. The Data Interchange Form must be logically capable of expressing the most sophisticated data base structure known, but be physically simplistic without being overly inefficient. Consequently herein lies a direction for future research.

REFERENCES

1. Fry, J. P. and D. W. Jeris, "Towards a Formulation and Definition of Data Reorganization," *Proc. 1974 ACM SIGMOD Workshop on Data Description, Access and Control,* R. Rustin (Ed.), Ann Arbor, Michigan, May 1974, pp. 83-100.
2. Bakkom, D. E. and J. A. Behymer, "Implementation of a Prototype Generalized File Translator," *Proc. 1975 ACM SIGMOD International Conf. on Management of Data,* W. F. King (Ed.), San Jose, California, May 1975, pp. 99-110.
3. Shoshani, A., "A Logical Level Approach to the Data Base Conversion," *Proc. 1975 ACM SIGMOD Conference,* W. F. King. (Ed.), San Jose, CA, May 1975, pp. 112-122.
4. Rameriz, J. A., "Automatic Conversion of Data Conversion Programs Using a Data Description Language," *Proc. 1974 ACM SIGFIDET Workshop on Data Description, Access and Control,* R. Rustin (Ed.), Ann Arbor, MI, May 1974, pp. 207-225.
5. Lewis, K. H., et al., "Translation Definition Language Reference Manual," Data Translation Project, The University of Michigan, 1976.
6. Taylor, R. W., "Generalized Data Base Management System Data Structures and Their Mappings to Physical Storage," PhD Dissertation, The University of Michigan, December, 1971.
7. Birss, E. W. and J. P. Fry, "A Comparison of Two Languages for Describing Stored-Data," Data Translation Working Paper #401, The University of Michigan, 1975.
8. Fry, J. P., R. L. Frank and E. A. Hershey, III, "A Developmental Model for Translation," *Proc. 1972 ACM SIGFIDET Workshop on Data Description, Access and Control,* A. L. Dean (Ed.), Denver, Colorado, November 1972, pp. 77-106.
9. Navathe, S. B. and J. P. Fry, "Restructuring for Large Data Bases: Three Levels of Abstraction," *ACM Transactions on Database Systems* (1:1), March 1976.
10. Hershey, E. A., III and P. W. Messink, "A Data Base Management System for PSA Based on DBTG 71," ISDOS Working Paper #88, ISDOS Research Project, The University of Michigan, 1975.
11. Birss, E. W., M. E. Deppe and J. P. Fry, "Research and Data Reorganization Capabilities of the Version IIA Data Translator," Data Translation Working Paper (no number), The University of Michigan, 1975.
12. CODASYL Systems Committee, *A Feature Analysis of Generalized Data Base Management System,* 1971.
13. Merten, A. G. and J. P. Fry, "A Data Description Approach to File Translation," *Proc. 1974 ACM SIGMOD Workshop on Data Description, Access and Control,* Ann Arbor, Michigan, May 1974, pp. 191-205.
14. Housel, B., V. Lum and N. Shu, "Architecture to an Interactive Migration System (AIMS)," *Proc. 1974 ACM SIGFIDET Workshop on Data Description, Access and Control,* R. Rustin (Ed.), Ann Arbor, MI, May 1974, pp. 157-169.
15. Housel, B., D. Smith, N. Shu and V. Lum, "Define: A Non-Procedural Data Description Language for Defining Information Easily," *Proc. of 1975 ACM Pacific Conference,* San Francisco, CA, April 1975, pp. 62-70.
16. SDDTTG, "An Approach to Stored-Data Definition and Translation," *Proc. 1972 ACM SIGFIDET Workshop on Data Description, Access and Control,* A. L. Dean (Ed.), Denver, CO, November 1972, pp. 13-55.
17. SDDTTG, "A Stored-Data Definition Language for the Translation of Data," The Stored-Data Definition and Translation Task Group of CODASYL Systems Committee, 1976, (Draft).
18. Smith, D. P., "An Approach to Data Description and Conversion," PhD Dissertation, The University of Pennsylvania, December, 1971.

Experiments with a symbolic evaluation system

by WILLIAM E. HOWDEN
University of California at San Diego
La Jolla, California

ABSTRACT

Symbolic evaluation techniques can be used to determine the cumulative effects of a program's calculations on the branching predicates and output variables in the program. If the evaluation techniques are carefully and selectively applied, they can be used to generate revealing symbolic representations of the computations carried out by the paths in a program, and of the systems of predicates that describe the input data that causes program paths to be executed. A symbolic evaluation system called DISSECT is described which can be used to analyze FORTRAN programs. The system includes a sophisticated command language that allows the user to selectively apply symbolic evaluation techniques to different program paths and subpaths. The command language allows the user to carry out different levels of symbolic testing of a program and to construct systems of predicates that can be used to automate the generation of numeric test data. Experiments with the system which illustrate its advantages and limitations are included. DISSECT can be used to carry out a systematic, documented reliability analysis of a program. The paper concludes with a discussion of the potential use of systems like DISSECT as the basic software certification tool in the software development process.

INTRODUCTION

When a program path is executed by running the program on a given input, the correctness of the path for that input can be determined by examining the effects of the calculations carried out by the path. If the path is executed "symbolically" rather than with actual data, it may be possible to use a single execution to illustrate its correctness on a large subset of the input domain rather than on just a single value. Symbolic execution of a program is carried out by giving dummy symbolic values rather than actual numeric (string, logical etc.) values to all or some of the input variables of the program. An expression in the program involving variables with symbolic values is evaluated by substituting the current symbolic values of the variables into the expression. The resulting expression is then simplified algebraically. All operators having only actual as opposed to symbolic operands are evaluated in the normal way. The resulting expression is the symbolic value of the original expression.

Figure 1 contains a program for carrying out polynomial interpolation.[1] The documentation for the program describes it as consisting of four segments, each of which computes part of the interpolation process. Lines 34 through 48 are supposed to compute a set of coefficients a_i which are given in terms of y_i and Δ_i by the formula:

$$a_k = \frac{y_k}{\prod_{i=1}^{k-1} (\Delta_k - \Delta_i)} - \sum_{j=1}^{k-1} \frac{a_j}{\prod_{i=j}^{k-1} (\Delta_k - \Delta_i)}$$

The program is written so that a_k corresponds to A(k), Δ_i to DELTA(i) and y_i to Y(I1+i−1). Symbolic values are designated by alphanumeric strings surrounded by quotes. Suppose "I1," "DELTA (I)" I=1, 10 and 2 are assigned to I1, DELTA(I) I=1,10 and NTERMS at statement 36 and "A(1)" to A(1) at statement 37. Then the effect of the calculations carried out in a single iteration of the loop in lines 34 through 47 can be determined by symbolically executing the code and printing out the values of A(1) at statement 36 and A(k) at statement 46. Figure 2 contains these symbolic values. The quotes designating symbolic values are deleted from the output to increase readability whenever this is unambiguous. Inspection of the values reveals that they agree with the formula and that this part of the program is correct for all input data having NTERMS=2.

Symbolic evaluation can be used to generate symbolic representation of the effects of the calculations carried out by paths in a program. It can also be used to generate sets of predicates in input variables which describe the input data that causes different program paths to be executed. A "complete" symbolic evaluation analysis of a program can be used as the validation documentation for the program.

Certain features must be present in an automated symbolic evaluation system in order for the system to be useful in analyzing realistic programs. The user

```
      SUBROUTINE INTERP(X, Y, NPTS, NTERMS, XIN, YOUT) #0
      DOUBLE PRECISION DELTAX, DELTA, A, PROD, SUM #1
      DIMENSION X(1), Y(1)                         #2
      DIMENSION DELTA(10), A(10)                   #3
C                                                  #4
C        SEARCH FOR APPROPRIATE VALUE OF X(1)      #5
C                                                  #6
   11 DO 19 I=1, NPTS                              #7
      IF (XIN-X(I)) 13, 17, 19                     #8
   13 I1=I-NTERMS/2                                #9
      IF (I1) 15, 15, 21                           #10
   15 I1=1                                         #11
      GO TO 21                                     #12
   17 YOUT=Y(I)                                    #13
   18 GO TO 61                                     #14
   19 CONTINUE                                     #15
      I1=NPTS-NTERMS+1                             #16
   21 12=I1+NTERMS-1                               #18
      IF (NPTS-I2) 23, 31, 31                      #19
   23 I2=NPTS                                      #20
      I1=I2-NTERMS+1                               #21
   25 IF (I1) 26, 26, 31                           #22
   26 I1=1                                         #23
   27 NTERMS=I2-I1+1                               #24
C
C        EVALUATE DEVIATIONS DELTA                 #26
C                                                  #27
   31 DENOM=X(I1+1)-X(I1)                          #28
      DELTAX=(XIN-X(I1))/DENOM                     #29
      DO 35 I=1, NTERMS                            #30
      IX=I1+I-1                                    #31
      DELTA(I)=(X(IX)-X(I1))/DENOM                 #32
   35 CONTINUE                                     #32.1
C                                                  #33
C        ACCUMULATE COEFFICIENTS A                 #34
C                                                  #35
   40 A(1)=Y(I1)                                   #36
   41 DO 50 K=2, NTERMS                            #37
      PROD=1.                                      #38
      SUM=0.                                       #39
      IMAX=K-1                                     #40
      IXMAX=I1+IMAX                                #41
      DO 49 I=1, IMAX                              #42
      J=K-I                                        #43
      PROD=PROD * (DELTA(K)-DELTA(J))              #44
   49 SUM=SUM-A(J)/PROD                            #45
      A(K)=SUM+Y(IMAX)/PROD                        #46
   50 CONTINUE                                     #47
C                                                  #48
C        ACCUMULATE SUM OF EXPANSION               #49
C                                                  #50
   51 SUM=A(1)                                     #51
      DO 57 J=2, NTERMS                            #52
      PROD=1.                                      #53
      IMAX=J-1                                     #54
      DO 56 I=1, IMAX                              #55
   56 PROD=PROD * (DELTAX-DELTA(I))                #56
   57 SUM=SUM+A(J)*PROD                            #57
   60 YOUT=SUM                                     #58
   61 RETURN                                       #59
      END                                          #60
```

Figure 1—Interpolation subroutine

```
A(1)=Y(I1)
A(2)=(Y(I1+1)/(DELTA(2)-DELTA (1)))
         -(A(1)/(DELTA(2)-DELTA (1)))
```

Figure 2—Symbolic values for A(1) and A(2)

must be able to easily set up and carry out a number of different analyses. He must be able to select subsegments of a program and individual paths or classes of paths. The system must contain facilities for assigning actual or symbolic values to variables and for printing out values of variables at different points in a program. The user may also wish to print out systems of predicates formed by symbolically evaluating the branch conditions in a path and to check the consistency of these systems.

The remaining sections of the paper describe a sophisticated symbolic evaluation system called DISSECT and a number of experiments using the system. DISSECT can be used to analyze programs written in ANSI Standard FORTRAN. In the experiments the system is used to analyze two complex statistical routines taken from Reference 1.

THE DISSECT SYSTEM

(a) Structure of DISSECT—The DISSECT system operates on two input files and produces an output file. One of the input files contains a FORTRAN program to be analyzed and the other contains the DISSECT commands which tell the system what kind of symbolic evaluation analysis to carry out. The output file contains the results of the analysis.

The command file for a DISSECT analysis is divided into a number of *cases*. The program in the input file is processed completely for each case. Each case has a section for a text description of the part of the program to be analyzed by the case. This text is not processed by the system and is considered to be the specification for the case, The rest of a case contains commands which identify the part of the program which corresponds to the case, commands which assign (symbolic) values to variables and commands which specify what output is to be generated.

The output file which is generated by a DISSECT analysis is also divided into cases. Each output case contains the case specifications and commands as well as the output generated by the system for that case. The user can check the validity of his program by comparing case specifications with system output for cases.

A program *path* is a possible flow of control through the program. A path is *feasible* if at least one element of the program's input domain causes that path to be executed. In general, a complete set of DISSECT cases for a program should "cover" the program in some sense. One approach is to analyze each feasible path (up to some number of iterations of loops). Complex programs having many paths can be divided into segments and analyzed using separate cases.

(b) DISSECT commands—The DISSECT commands can be divided into three groups: path selection, output and value commands. The commands can either be given in the *global commands section* of a DISSECT command file or can be given as part of a case. If they appear in the global commands section they apply to all cases. The commands can be used with or without a statement sequence number. In general, when used with a sequence number they are executed only when the system is evaluating that statement. Otherwise they are used for every statement or, in some cases, are only applied at the end of the DISSECT analysis of a program path.

(c) Path Selection Commands—DISSECT processes a case by symbolically evaluating one or more paths in a program. The path selection commands cause DISSECT to select part of a program for analysis by directing it to follow certain paths through a program. The "SELECT" command is used for directing DISSECT to follow a specified branch or branches from a conditional branching statement. The "LOOP" command directs DISSECT to carry out a given number of iterations of a loop. Loops are specified by naming the statement number of the first statement in the loop.

Examples

(i) n SELECT .GT. will cause DISSECT to select the .GT. branch from an arithmetic conditional statement n.

(ii) n SELECT ALL will cause DISSECT to set up a subpath for each branch from conditional statement n.

(iii) n LOOP k will cause DISSECT to iterate loop n k times.

(iv) n SKIP m will cause DISSECT to skip from statement n to m during processing of a program. SKIP can be used to set up cases which only deal with paths through segments of a program rather than the whole program.

(d) Output commands—The user of DISSECT can generate several different kinds of output. The PATH command causes the system to print out the sequence numbers of the statements in the paths in a case. PATH DESCRIPTION will cause the output of all of the statements in the paths. PREDICATES will result in the construction of symbolically evaluated systems of predicates which describe the input that causes paths to be followed. The OUTPUT command can be used to print out the symbolic values of variables, symbolically evaluated subroutine calls and symbolically evaluated program output statements. If an output command is preceded by a statement sequence number then the command is invoked when the system encounters that statement during its symbolic evaluation of the program. If the command is not preceded by a statement number then, in general, it is invoked after the completion of any path belonging to the case containing the output command.

(e) Value commands—The most important value assigning command in DISSECT is the ASSIGN command. ASSIGN can be used to assign either actual or symbolic values to any variable at any point in the processing of a program. The DISSECT system is designed so that symbolic values are automatically assigned to variables whenever the variables appear in SUBROUTINE headers or COMMON or input statements. The automatically assigned symbolic values are text strings formed from the variable names. ASSIGN can be used to override these default symbolic value assignments or to assign actual values.

ASSIGN is used in basically two different ways. In many situations, the user will want to carry out a symbolic evaluation of a path in which some of the variables are given actual values. He can use ASSIGN to give these values to the variables at the beginning of the path. In other situations a complicated segment of calculations on several variables divides naturally into several segments and a user will want to print out the symbolic values of one or more variables at the end of one segment and then reset the values of the variables to simple symbolic values before continuing with the processing of the next segment. The ability to do this avoids the necessity of having to analyze complicated symbolic expressions resulting from the symbolic execution of several conceptually distinct sequences of operations.

(f) Conditional Execution of commands—There are two ways in which a user can specify that a DISSECT command is to only be applied under certain conditions. The first involves the use of a *conditional form.* If a command appears in the form "IF condition THEN command 1 [ELSE command 2]" then command 1 is only carried out if the condition holds (and command 2 if the condition does not hold). Conditional forms are not commands and cannot be nested.

Conditions are constructed using three types of expressions. The first consists of ordinary program expressions in program variables. The second consists of the special variable ATTRIBUTE and the third the special function LOOPCOUNT(n). DISSECT contains a number of commands for attaching *attributes* to paths during their symbolic evaluation. It is possible to specify that the execution of a DISSECT command is conditional upon association of a given attribute with a path. LOOPCOUNT(n) returns an integer giving the number of times that loop n has been iterated by the path currently being traversed. When the function is called it is assumed that the statement currently being processed by DISSECT is in the loop and that LOOPCOUNT(n) is the number of iterations that have been completed during the current traversal of the loop.

The second way of conditionally carrying out a DISSECT command involves the use of the CONSISTENCY option. Many of the DISSECT commands can be used with several flags. When the CONSISTENCY flag is attached to a command the system constructs the predicate that would be added to the system of predicates for the current path if the command were to be executed. If the addition of this predicate to the system would cause the system to be inconsistent then the command is not executed. The CONSISTENCY option can be used to stop DISSECT from traversing program paths that would result in the generation and analysis of infeasible program paths. The consistency routines which are currently implemented are very simple. Although they will catch only certain kinds of inconsistencies they have proved to be very powerful.

A related feature in DISSECT is the DEFAULT option. DISSECT expects that certain kinds of program statements will always have a command associated with them. It expects the command file to contain, for example, path selection commands for each conditional branching statement which it encounters when it is processing a program path. In general, a user may only want to construct commands for a fraction of the branching statements in a program. He can "cover" the remaining branching statements by constructing selection commands which have no statement number and which include the DEFAULT option. When DISSECT reaches a conditional statement it first looks to see if there are any selection commands for the statement which do not have the DEFAULT option set. If there are none it then tries to find a selection command that it can apply which has the DEFAULT option set. Recall that DISSECT commands which do not have statement numbers are applied by DISSECT to all appropriate statements during the processing of the program.

EXAMPLES

(a) Interpolation Example—The first example describes the use of DISSECT in analyzing the interpolation program in Figure 1. In both this and the next example DISSECT was used to confirm that the program agreed with its specifications. The INTERP routine is written so that the number of points NTERMS used in the interpolation process may be less than the number of points available (NPTS). The first segment of the routine, lines 1-25, decides which points to use. It involves choosing a value for I1. The points X(I1), X(I1+1), . . ., X(I1+n−1) in the program correspond to the points x_1, x_2, . . ., x_n in the documented formulae (n=min{NPTS, NTERMS}). The documentation states that x_1 (i.e. X(I1)) "is chosen so that x_1 and x_n straddle x". "If the value of x is too close to the lower or upper limit of the values of x_i, the corresponding value of x_1 or x_n is set equal to the limiting value."

The documentation for this segment of the program is quite vague. A casual reading of the program reveals that the segment is of some complexity and has a number of paths. The process that is to be carried out by the part of the program appears to be typical of the types of processes that may not work for limiting values in the input. It was decided to examine the paths through this section of code for NPTS=1 and 2 (the limiting cases) and also for NPTS=3. The DISSECT command file for those three cases is given in Figure 3.

These three cases cause DISSECT to analyze all paths in the program up to statement 28 for NPTS=1, 2 and 3. Case 1 has 10 paths, case 2 17 paths and case 3 24 paths. This is a large number of paths but it was found that the output for each was easy to read and that it was easy to determine if the program was correct for the case. The output for each case is divided into subcases, several of which are reproduced in Figure 4. Each subcase corresponds to a path.

The structure of the code in the first segment of the program is such that the complete set of paths for cases one, two and three indicates that the segment is correct. Note that we have not formally proved that the segment is correct. The symbolic predicates and values which are produced assist the user in reading the code and in carrying out a proof which is partly formal and partly informal.

The second segment of DISSECT, lines 26-33 is supposed to compute the following values of Δ and Δ_i.

$$\Delta=\frac{x-x_1}{x_2-x_1} \qquad \Delta_i=\frac{x_i-x_1}{x_2-x_1}$$

The correctness of this segment is easily confirmed with two or three simple cases. The commands for the case where NPTS=2 are given in Figure 5.

The output for this case is given in Figure 6. In the program Δ is represented by DELTAX. In reading the output recall that x_i is represented by $X(I1+i-1)$.

The Case 6 command ASSIGN I1="I1" ensures that the output for Case 6 will use the symbol I1 to stand for I1 rather than the value it may have as the result of earlier calculations.

Symbolic output for the code in the third segment, lines 34 through 48, appears in Figure 2. The segment was thoroughly examined by looking at the output for the cases where NPTS=3 and 4.

The final segment of the program, lines 49 through 60, is supposed to compute the formula:

$$y(x)=a_1+\sum_{j=2}^{n}\left[a_j\prod_{i=1}^{j-1}(\Delta-\Delta_i)\right].$$

This segment can be checked by constructing cases corresponding to n=1, 2 and 3. The case containing the commands for n=2 appears in Figure 7. The program uses the variable YOUT to represent the value y(x). The output for the case appears in Figure 8.

(b) Correlation Example—The complexity of the INTERP routine is due both to its control logic and its array manipulation operations. DISSECT was useful in analyzing INTERP by allowing the user to look at how the program operated for arrays of given dimensions.

In the correlation example, DISSECT was used to

```
TITLE: ANALYSIS OF INTERP
GLOBAL COMMANDS:
   MAXPATHS 300;
   MAXLENGTH 200;
   DEFAULT LOOP ANY CONSISTENT (1-10);
   DEFAULT SELECT ANY CONSISTENT;
   PATH; PREDICATES;

CASE 1: ANALYSIS OF SEGMENT OF CODE THAT DETERMINES I1.
   SET NPTS=1
CASE COMMANDS:
   OUTPUT I1, NTERMS;
   SELECT ALL;
 7 ASSIGN NPTS=1;
28 HALT;

CASE 2; SET NPTS=2.
CASE COMMANDS:
   OUTPUT I1, NTERMS;
   SELECT ALL;
 7 ASSIGN NPTS=2;
28 HALT;

CASE 3: SET NPTS=3.
CASE COMMANDS:
   OUTPUT I1, NTERMS;
   SELECT ALL;
 7 ASSIGN NPTS=3;
28 HALT;
```

Figure 3—DISSECT commands for first segment of INTERP

```
CASE 1.1
PATH:  0-12 18-24

PREDICATES:
  : 1   0  SUBROUTINE INTERP(X,Y,NPTS, NTERMS, XIN, YOUT)
  : 7   8  XIN-X(1) .LT. 0
  : 9  10  1-(NTERMS/2) .LE. 0
  :13  19  1-NTERMS .LT. 0
  :16  22  2-NTERMS .LE. 0

OUTPUT:
  :18  28  I1=1
           NTERMS=1

CASE 2.3
PATH:  0-12 18-19

PREDICATES:
  : 1   0  SUBROUTINE INTERP(X,Y,NPTS, NTERMS, XIN, YOUT)
  : 7   8  XIN-X(1) .LT. 0
  : 9  10  1-(NTERMS/2) .LE. 0
  :13  19  2—NTERMS .GE. 0

OUTPUT:
  :13  28  I1=1
           NTERMS=NTERMS

CASE 3.11
PATH:  0-8 15 7-10 18-24

PREDICATES:
  : 1   0  SUBROUTINE INTERP(X,Y,NPTS, NTERMS, XIN, YOUT)
  : 7   8  XIN-X(1) .GT. 0
  :11   8  XIN-X(2) .LT. 0
  :13  10  2-(NTERMS/2) .GT. 0
  :15  19  2+NTERMS/2-NTERMS .LT. 0
  :18  22  4-NTERMS .LE. 0

OUTPUT:
  :20  28  I1=1
           NTERMS=3
```

Figure 4—DISSECT output for first segment of INTERP

```
CASE 6: TEST SECOND SEGMENT WITH NPTS=4.
CASE COMMANDS:
     OUTPUT, DELTAX, DELTA;
   7 SKIP 28;
  28 ASSIGN NTERMS=4;
  28 ASSIGN I1="I1";
  36 HALT
```

Figure 5—Case commands for symbolic evaluation of second segment

```
CASE 6.1
PATH 0-3 28-32.1 30-32.1 30-32.1 30-32.1 30

PREDICATES:

OUTPUT:
        : 1   0  SUBROUTINE INTERP(X,Y, NPTS, NTERMS, XIN, YOUT)
        : 6  28  **ASSIGN I1=I1
        :29  36  DELTAX=(XIN-X(I1))/(X(I1+1)-X(I1))

                 DELTA(4)=(X(I1+3)-X(I1))/(X(I1+1)-X(I1))
                 DELTA(3)=(X(I1+2)-X(I1))/(X(I1+1)-X(I1))
                 DELTA(2)=(X(I1+1)-X(I1))/(X(I1+1)-X(I1))
                 DELTA(1)=(X(I1)-X(I1))/(X(I1+1)-X(I1))
```

Figure 6—Symbolic output for analysis of second segment

```
CASE 9: ANALYSIS OF SEGMENT FOR COMPUTING FINAL VALUE OF Y.
        NTERMS=2.

CASE COMMANDS:
    OUTPUT YOUT;
 7  SKIP 51;
51  ASSIGN NTERMS=2, A="A", DELTAX="DELTAX";
51  ASSIGN DELTA="DELTA";
```

Figure 7—Case commands for analyzing last segment of program

analyze a program whose complexity is entirely due to its control logic and to the computations it carries out. The program has no arrays and is not complicated by looping mechanisms for carrying out array operations.

The PCORRE routine is used to determine the probability $P_c(r,N)$ that N random data points would yield a linear-correlation coefficient as large or larger than an observed correlation value $|r|$. The documentation for PCORRE lists two formulae which are supposed to be computed by the routine. For $v=N-2$, one of the formula is for v even and the other for v odd. The formula for v even is

$$P_c(r,N)=1-\frac{2}{\sqrt{\pi}}\,\frac{\Gamma[(v+1)/2]}{\Gamma(v/2)}\left\{\sum_{i=0}^{I}\left[(-1)^i\,\frac{I!}{(I-i)!i!}\,\frac{|r|^{2i+1}}{2i+1}\right]\right\}$$

where $I=\frac{v-2}{2}$ and Γ is the gamma function. The formula for v odd is equally complex.

The PCORRE routine divides naturally into three segments. The first determines the quantity v and whether it is even or odd. The second and third compute $P_c(r,N)$ for even and odd v. The part of the code that computes $P_c(r,N)$ for v even is reproduced in Figure 10.

A command file containing the case in Figure 11 was constructed for analyzing PCORRE. The output for one of the subcases which analyzes the first two segments of the routine is reproduced in Figure 12.

The text strings in quotes after some of the commands in Figure 11 are attributes. Each time a path follows a branch associated with a particular SELECT command, the path is assigned any attributes listed for that branch. The complete collection of attributes for a path is printed along with the output for the path. The use of attributes makes it easy to identify paths in terms of particular properties associated with the branches in the paths.

```
CASE 9
PATH: 0-3 51-56 55 57 52 58-59
PREDICATES:
OUTPUT:
 : 6  51  **ASSIGN A=A
 : 7  51  **ASSIGN DELTAX=DELTAX
 : 8  51  **ASSIGN DELTA=DELTA
 :21  59  YOUT=A(1)+A(2) * (DELTAX-DELTA(1))
```

Figure 8—Output for Case 9

It is evident from this output that the formula is correct for the case where v is even and $(v-2)/2=1$ DISSECT was also used to generate symbolic out for the cases where $(v-2)/2=2$ and $(v-2)/2=3$. This output taken together with the pattern of the code in the program, indicates that the program computes the correct formula for v even. DISSECT was also used to generate symbolic output for the code which calculates $P_c(v,N)$ for v odd.

RELATED WORK

The DISSECT system is built on an earlier path analysis system[2] which can be used to generate descriptions of the sets of input data that cause classes of program paths to be executed. The major improvement is the addition of a command language which allows the user to selectively control the application of the analysis routines. In addition, DISSECT has many features not present in the previous system.

Several systems have been constructed which can be used to carry out program analysis similar to those which can be carried out with DISSECT. The EFFIGY,[3] SELECT,[4] RXVP[5] and the system described by Clarke in Reference 6 all allow the selection and evaluation of paths in computer programs. The DISSECT system is closest to the EFFIGY and SELECT systems. Symbolic evaluation can also be used in constructing program proofs.[7,8]

DISSECT is unique in its use of cases to structure the validation analysis of a program. It is also the only system that includes a language that allows a user to write simple analysis procedures (the command language). The EFFIGY and SELECT systems are path-following procedures which the user directs by selecting conditional statement branches interactively.

CONCLUSIONS

In the experiments with DISSECT which are described above, the user was faced with analyzing and determining the validity of a program he had not seen before. He was provided with the program and documentation and no outside help. The pattern of usage of DISSECT was one in which the user alternated between reading a program and running DISSECT. First the program and the documentation were studied and a preliminary command file constructed. The

```
      FUNCTION PCORRE (R,NPTS)                                     #1
      DOUBLE PRECISION R2, TERM, SUM, FI, FNUM, DENOM              #2
C                                                                  #3
C       EVALUATE NUMBER OF DEGREES OF FREEDOM                      #4
C                                                                  #5
   11 NFREE=NPTS-2                                                 #6
      IF (NFREE) 13, 13, 15                                        #7
   13 PCORRE=0.                                                    #8
      GO TO 60                                                     #9
   15 R2=R**2                                                      #10
      IF (1.-R2) 13, 13, 17                                        #11
   17 NEVEN=2*(NFREE/2)                                            #12
      IF (NFREE-NEVEN) 21, 21, 41                                  #13
C                                                                  #14
C       NUMBER OF DEGREES OF FREEDOM IS EVEN                       #15
C                                                                  #16
   21 IMAX=(NFREE-2)/2                                             #17
      FREE=NFREE                                                   #18
   23 TERM=ABS (R)                                                 #19
      SUM=TERM                                                     #20
      IF (IMAX) 60, 26, 31                                         #21
   26 PCORRE=1.-TERM                                               #22
      GO TO 60                                                     #23
   31 DO 36 I=1, IMAX                                              #24
      FI=I                                                         #25
      FNUM=IMAX-I+1                                                #26
      DENOM=2*I+1                                                  #27
      TERM=-TERM * R2 * FNUM/FI                                    #28
   36 SUM=SUM+TERM/DENOM                                           #29
      PCORRE=1.128379167 * (GAMMA((FREE+1.)/2.)/GAMMA(FREE/2.))    #30
      PCORRE=1.-PCORRE*SUM                                         #31
      GO TO 60                                                     #32
```

Figure 10—First two segments of PCORRE

```
CASE 1: MINIMUM CASE TO ILLUSTRATE THE PCORRE FORMULAE.
CASE COMMANDS:

         OUTPUT PCORRE
7        ASSIGN NFREE="V";
7,11     SELECT .GT.;
13       SELECT .GT. "DEGREES OF FREEDOM ODD.";
13       SELECT .LE. "DEGREES OF FREEDOM EVEN";
21,39    SELECT .LT. "SUMMATION MAX NEGATIVE—NO CALCULATION";
21,39    SELECT .EQ. "SUMMATION MAX ZERO—NO SUMMATION";
21,39    SELECT .GT. "SUMMATION MAX POSITIVE";
```

Figure 11—Case commands for analyzing PCORRE

```
CASE 1.1
ATTRIBUTES: SUMMATION MAX POSITIVE. DEGREE OF FREEDOM EVEN

PATH: 1-7 7 10-21 24-29 24 30-32 49

PREDICATES:

  : 1   1  FUNCTION PCORRE(R,NPTS)
  : 5   7  V .GT. 0
  : 7  11  1.0-R**2 .GT. 0
  : 9  13  V-2*(V/2) .LE.0
  :14  21  ((V-2)/2) .GT. 0
  :21  24  ((V-2)/2).GE.1

OUTPUT:

  : 1   1  FUNCTION PCORRE (R,NPTS)
  :26  49  PCORRE=1.0-1.128379*(GAMMA((1.0+V)/2.0)/
                     GAMMA(V/2.0))*ABS(R)+
                     1.128379*(GAMMA((1.0+V)/2.0)/
                     GAMMA (V/2.0)) * ((ABS(R)*R**2*((V-2)/2))/3)
```

Figure 12—Symbolic output for case 1.1 of analysis of PCORRE

command file was based on an initial understanding of the structure of the program. The output which was generated by DISSECT from the preliminary command file was often inadequate. In some cases, too many paths were generated for a piece of code. In other cases, more analysis of a piece of code, or analysis with different actual or symbolic assignments of variables, was needed. A second DISSECT analysis resulted in a better understanding of the program which sometimes prompted the user to carry out a third analysis.

The readability and usefulness of DISSECT output can be judged from the above examples. The output was generated from either the first or second command files which were constructed in the process of analyzing the programs. One or two additional versions of the command file would produce what would probably be the final version.

It is our experience that systems like DISSECT can be useful in two ways: The first is the help that the system can give the user in carrying out a validation analysis of a program. The output from a symbolic evaluation of a piece of code is often much more revealing than output from an execution with actual data. Similarly, the symbolic evaluation of a system of predicates associated with a path provides documentation describing the input associated with the path.

DISSECT can be useful in helping to "unravel" the computations carried out by different types of programs. Some programs are complicated to read because they are cluttered with the control structure needed to carry out an operation iteratively over the elements of a data structure. The validity of these programs can often be checked by looking at the manipulations that are carried out for structures of fixed sizes. DISSECT can be used to generate descriptions of these manipulations. Other programs are complicated to read because they contain control structure for constructing, and at the same time computing a value for an iterative formula. The number of iterations used in constructing the formula is often dependent on an input variable. DISSECT can be used to generate the instances of this formula that are computed by the program for different numbers of iterations. These instances, together with a knowledge of the structure of the iteration used by the program are often enough to convince the user that the program is correct. Examples 1 and 2 illustrate this use of DISSECT for programs of this type.

The second way in which DISSECT can be useful is in forcing a systematic, intuitively meaningful validation discipline on the programmer. DISSECT can be used as the basis of a validation methodology for a verification group. In verifying a program, the user must first break the specifications for the program down into cases. Each case is then described in terms of a case specification. The parts of the program which are supposed to take care of these cases are identified using the command language. The system carries out the specified analysis and the user compares the output with the case specifications. After several rounds of improving the cases and possibly correcting program errors, a full validation document is produced. Experience may indicate that other types of validation analysis facilities should be added to the DISSECT without changing the basic structure of the system.

There are several situations in which systems like DISSECT may fail or not be useful. If a program is in error because some case was not coded into it, and no DISSECT case is constructed to correspond to this forgotten case, then the error may not be discovered. It is suspected that DISSECT may not be as useful for assembly language programs as it is for high-level language numeric programs. Since, in theory, a well-structured program is self-documenting and it is possible to understand the code by reading it, it might be argued that DISSECT will only be useful for programs written in languages like FORTRAN. This is only partly true. The observations made above about program complexity due to the mixing together of control structure for manipulating data structures and the computations to be carried out on the structures are true for most programming languages. The same thing is true of complexity due to mixing together the code which constructs an iteratively defined formula together with the computations for evaluating the formula.

Further experiments will be carried out with DISSECT and improvements to the system are planned. The system will be used to analyze programs known to contain errors in order to determine its usefulness in detecting bugs. Use of the system in a full-scale software development project is also being planned.

ACKNOWLEDGMENTS

The DISSECT system was programmed by the author and R. Hoffman. Mr. Hoffman is primarily responsible for the symbolic evaluation and output routines.

Part of the algebraic simplifier in the output routine was derived from a program which is part of the SELECT system.[4] The use of the program is gratefully acknowledged.

The research described in the paper was carried out as part of a McDonnell Douglas Astronautics project in program testing which is funded by the National Bureau of Standards. The project is supervised by Z. Jelinski and Leon Stucki of McDonnell Douglas. Mr. Stucki cooperated in the design of the DISSECT command language.

REFERENCES

1. Benington, Philip R., *Data Reduction and Error Analysis for the Physical Sciences*, McGraw-Hill, 1969.

2. Howden, William E. and Jeffrey Laub, "Automatic Case Analysis of Programs," *Proceedings of Computer Science and Statistics: 8th Annual Symposium on the Interface*, Los Angeles, February, 1975.
3. King, James C., "A New Approach to Program Testing," *Proceedings of the International Conference on Reliable Software*, Los Angeles, April 1975.
4. Boyer, Robert S., Bernard Elspas and Kan N. Levitt, "SELECT—A Formal System for Testing and Debugging Programs by Symbolic Execution," *Proceedings of the International Conference on Reliable Software*, Los Angeles, April 1975.
5. Miller, E. F., "RXVP: An Automated Verification System for FORTRAN," *Proceedings of Computer Science and Statistics: 8th Annual Symposium on the Interface*, Los Angeles, February 1975.
6. Clarke, Lori, *A System to Generate Test Data and Symbolically Execute Programs*, Department of Comp. Science, University of Colorado, CU-CS-060-75, February 1975.
7. Deutsch, L. P., *An Interactive Program Verifier*, Ph.D. dissertation, University of California, Berkeley, May 1973.
8. Burstall, R. M., "Proving Correctness as Hand Simulation with a Little Induction," *Proceedings of IFIPS 74*, North Holland Publishing Company, 1974.

Some experience with DAVE — A Fortran program analyzer*

by LEON J. OSTERWEIL and LLOYD D. FOSDICK
University of Colorado
Boulder, Colorado

ABSTRACT

This paper describes DAVE, an automatic program testing aid which performs a static analysis of Fortran programs. DAVE analyzes the data flows both within and across subprogram boundaries of Fortran programs, and is able to detect occurrences of uninitialized and dead variables in such programs. The paper shows how this capability facilitates the detection of a wide variety of errors, many of which are often quite subtle. The central analytic mechanism in DAVE is a depth-first search procedure which enables DAVE to execute efficiently. Some experiences with DAVE are described and evaluated and some future work is projected.

INTRODUCTION

There is currently a great deal of interest in creating systems capable of assisting in the development of error-free programs. This interest results both from an awareness that erroneous programs are expensive and potentially lethal and from the fact that the problems involved in producing error-free programs are challenging and stimulating. As might be expected in the case of such a problem, which has enormous economic significance and high intellectual appeal, the approaches to its solution are numerous and diverse. This diversity is shown by the following list of approaches, which is intended to be indicative and not exhaustive:

- Devise error resistant design and coding practices: The terms Structured Programming,[1] Stepwise Refinement,[2] and Top-Down Design[3] are often associated with work in this area.
- Create error resistant languages: Such investigators as Wirth[4] and Gannon and Horning[5] have identified error-prone language features and proposed languages which avoid them.
- Devise better organizational strategies for programming: The Chief Programmer Team strategy of Mills and Baker[6,7] is notable in this area.
- Prove the correctness of programs: This is a relatively difficult and time consuming process, which has been successful largely for relatively small programs. Current work,[8] however, offers hope that machine aids may eventually facilitate program proving for large programs as well.
- Build automated program testing aids: These aids can do such things as monitor program execution,[9,10,11] perform static diagnostic scans,[12,13] and help generate test data.[13,14]

It seems clear that in the future the results of work in several of these areas will be coordinated in any effort to produce high quality, error resistant programs. We feel certain, however, that because humans will always have faulty memories, be prone to commit keyboard errors, and will inject various other errors into their programs, that any such coordinated attack will surely include a testing activity. This activity should rely heavily upon automated program test aids. In addition, we feel that automated test aids are of particular importance at present, because they, unlike most of the other current approaches, offer some hope of helping determine the validity and worth of some of the enormous body of programs already in existence.

For these reasons, we created DAVE, an automated program testing aid which, we believe, embodies important new diagnostic capabilities.

DAVE AS AN AUTOMATED TESTING AID

DAVE performs a diagnostic scan of an ANSI Standard[15] Fortran program for the purpose of detecting erroneous or suspicious situations. Systems such as this are often referred to as static analysis systems, in that they do not require that the program be executed. As a consequence, their analytic results are not restricted in their applicability to a single execution. On the contrary, in DAVE's case it is possible to simulate in a limited way the effect of executing all sequences of statements in a program. Hence DAVE is able not only to detect errors, but, more important, it is also able to determine the absence of certain types of errors or suspicious situations for all possible executions of the

* This work supported by NSF Grants GJ-36461 and DCR75-90072.

program. Because of this latter capability DAVE is a valuable tool in examining existing programs for the purpose of validating them.

Should DAVE detect an error or suspicious situation along some execution sequence through the program an error or warning message describing the situation is produced. A human analyst must then determine the true importance of the message. At this point a dynamic analysis system might be used to instrument the program and gather detailed information about the progress of an actual execution of the sequence of statements which DAVE had pinpointed. Hence in this way DAVE is also useful as a debugging aid during program development.

ERRORS DETECTED BY DAVE

All program testing aids are incapable of determining whether or not a program is completely correct. A program testing aid can at best determine whether or not a program adheres to some specified standards. A violation of such a standard may be taken to be an a priori error or a suspicious condition, symptomatic of some other error. Hence in all program testing aids there must be an initial understanding of the standards against which programs are to be measured. In DAVE these standards all relate to the correct flow of data through a program. It is our contention that in a correctly executing program two rules should always be obeyed:

1. No variable will be used in a computation (referenced) until it has previously been assigned a value (been defined).
2. A variable, once defined, will subsequently be referenced before the variable is redefined or the program terminates.

DAVE's diagnostic scan determines whether either of these two rules can be violated for any sequence of statement executions. A violation of the first, called a type 1 anomaly, is a violation of the ANSI Fortran Standard[15] and is considered to be an a priori error. A violation of the second, called a type 2 anomaly, is considered to be a symptom of some other error. DAVE is able to detect a type 1 anomaly for any possible execution sequence. Thus if DAVE does not detect such an anomaly then none exists within the program. Hence DAVE is able to both detect the presence, and assure the absence of data flow anomalies. The former capability we refer to as error detection and the latter we refer to as validation. Clearly the foregoing implies that DAVE is able to validate programs for the absence of uninitialized variables.

In practice we have found, however, that anomalies of both types are usually symptoms of other errors. We have been gratified to find that the range of errors symptomatized by type 1 and type 2 anomalies is quite large, extending from misspellings to subprogram invocation errors. Because of this phenomenon of anomalies occurring as symptoms of other errors, it has turned out that DAVE has been most useful in indirectly detecting errors other than uninitialized and dead variables (in the sense of Reference 16).

More specifically, a large measure of DAVE's indirect error detection capability arises from the fact that DAVE performs its data flow analysis across subprogram boundaries. This data flow from one program unit to another must be completely determined if all possible anomalies are to be detected. Having made this complete determination, however, DAVE is in a position to also detect a variety of program unit communication errors such as illegal side effects and inconsistent COMMON declarations. Because this interprocedural data flow is often quite subtle, errors involving it are likewise often subtle and difficult for a human to identify. Hence it is not surprising that DAVE's error detection capabilities in this area have proven to be among its most useful features.

AN EXAMPLE

Figure 1 shows a somewhat contrived Fortran program which is designed to illustrate some of the error detection capabilities referred to in the previous section. The purpose of the program is to compute and

```
      COMMON /B/ AREA, COST
      READ (5,1) PSF, LCRT, D1, D2
      PI=3.1416
      IF (LCRT .NE. 1) GO TO 10
      AREA=AREAR (D1, D2)
      GO TO 100
10    IF (LCRT .NE. 2) GO TO 20
      AREA=AREAC (P, D1)
      GO TO 100
20    CALL AREAT (D1, D2, AREA)
100   CALL DOLS (PSF)
      WRITE (6, 2) COST
      STOP
1     FORMAT (F6.2, I2, 2F10.4)
2     FORMAT (1H, F8.2)
      END
      FUNCTION AREAR (A, B)
      AREAR=A * B
      RETURN
      END
      FUNCTION AREAC (PI, RAD)
      AREAC=PI * RAD ** 2
      RETURN
      END
      SUBROUTINE AREAT (B, H, AREA)
      AREAT=0.5 * B * H
      RETURN
      END
      SUBROUTINE DOLS (PSF)
      COMMON /B/ COST, AREA
      COST=PSF * AREA
      RETURN
      END
```

Figure 1—A program illustrating some of the error detection capabilities of DAVE

print out the cost of covering an area with some covering material. The program reads in PSF, the cost per square foot of the material; LCRT, an integer used to denote whether the area is a rectangle (if LCRT is 1), a circle (if LCRT is 2), or a triangle (if LCRT is 3); and D1 and D2, the two dimensions of the area (D2 is unused if LCRT is 2). The program then branches on LCRT to three different subprograms, AREAR, AREAC, and AREAT, which are supposed to compute the area of the rectangle, circle or triangle (respectively), and place the value of this area in the variable AREA. Subroutine DOLS is then called to compute COST, the product of AREA and PSF. Finally COST, the desired result, is printed out.

Close inspection of the program reveals that it contains errors, some of which are not very obvious. Perhaps the most obvious error is that the value of pi is set into the variable PI, but the variable P is used to pass this value into AREAC, the subprogram which requires it. A second error is that there is a misspelling in the subroutine AREAT. The third parameter is named AREA, but the body of the subroutine defines a value for the variable AREAT instead. Hence upon return there is no value given to the main program variable AREA, which is referenced in a subsequent computation. A third error involves the COMMON block B, which is used for communication between the main program and DOLS. B contains the variables AREA and COST. DOLS, which expects AREA to contain the computed area, uses it to compute the value of COST, which is then passed through B back to the main program. Unfortunately, the order of declaration of AREA and COST in the main program is the reverse of the order of declaration in DOLS.

Detection of these three errors would most likely be at least tedious using conventional debugging methods. The third error would cause any execution to be erroneous, but each of the first two would cause an erroneous execution only for a single specific value of LCRT. Hence it is reasonable to expect that the three errors would be ferreted out one at a time, perhaps with some difficulty, if the usual procedure of running test cases was followed.

DAVE would facilitate the detection of all three errors in only one diagnostic scan because each one causes data flow anomalies. In the case of the first error, DAVE would identify the definition of PI without subsequent reference as a type 2 anomaly. DAVE would also determine that the first argument in any invocation of AREAC must carry in a value. Hence in analyzing the main program DAVE would conclude that the invocation of AREAC would cause a type 1 anomaly, and would print an appropriate message. DAVE is unable to state directly the true error—namely a misspelling. The two anomaly messages, however, point strongly to the true error.

The second error, also a misspelling, is likewise strongly indicated by anomaly messages. In analyzing AREAT, DAVE would discover that the local variable AREAT is never referenced after definition, and print a message describing this type 2 anomaly. DAVE would also determine that the parameter AREA is neither referenced nor defined in the subroutine. This is regarded as a suspicious situation, and DAVE would produce a message describing it. Finally, in analyzing the main program, DAVE would discover that there is a sequence of statements leading up to the invocation of DOLS which does not cause the variable AREA to be defined (namely the one which includes the invocation of AREAT). No anomaly message will be printed because, as shall be seen, the third error causes DOLS to make no use of AREA. Had the third error not been present, however, a type 1 anomaly message would have been printed. In this case the interaction of two errors causes the suppression of one anomaly message. DAVE, nevertheless, produces two other messages in reponse to the second error.

The third error is a transposition of variables in a COMMON statement. DAVE would analyze DOLS and determine that it requires a value to be passed in through the second variable in COMMON block B, and that it passes out a value through the first variable in B. Upon analyzing the main program DAVE would find that COST, the second variable in COMMON block B, is never initialized before the invocation of DOLS—hence a type 1 anomaly message would be printed. DAVE would also discover that AREA, the first variable in block B, generally has a freshly computed value when DOLS is invoked. DAVE would observe the DOLS resets this value before it is ever referenced and print a type 2 anomaly message. Finally, DAVE would observe that AREA, the first variable in block B, is never referenced after its definition in DOLS and print another type 2 anomaly message. Here too, it is clear that these three messages strongly illuminate the transposition error, although it is never explicitly identified.

This brief example is intended to give an impression of how DAVE's analysis can assist in isolating subtle errors. We expect that the reader can see how DAVE is also useful in detecting other errors such as transposed statements, illegal side effects, and mismatched argument and parameter lists. Likewise the reader should be able to see that the use of an automatic aid such as DAVE is far more necessary in analyzing a large, complex, real-world program than in detecting the errors in this small, simple, contrived example.

THE DESIGN OF THE DAVE SYSTEM

DAVE performs its analysis by passing over the program units of a program, from the lowest level subprograms upward to the main program, analyzing each program unit exactly once, employing a depth-first search of a labelled flow graph of the program unit. Details of the system's design and implementation can be found in References 17 and 18, and hence are omitted here. For completeness, however, a brief simplified overview shall now be given.

DAVE's analysis is performed on labelled flow graphs, where a different graph represents each of the program units of the program. The nodes of a flow graph represent the program unit's statements and the edges represent intra-program-unit control transfers. Each node's label describes which variables are defined and referenced during the execution of the code corresponding to the node. These graphs are constructed at the start of DAVE's analysis. The graphs are easily constructed, but they cannot immediately be completely labelled, due to the undeterminable status of variables which are used as arguments to subprograms. Hence the graphs are left only partially labelled until a later phase of the analysis. All invocations of subprograms which are made by a program unit are carefully noted, however. After the last program unit flow graph has been created and partially labelled, the totality of these invocations is used to construct the program call graph, a graph whose nodes represent the program units and whose edges represent the subprogram invocations. Due to the impossibility of recursive calling chains in ANSI Standard Fortran, it is expected (although not always true) that the call graph will be acyclic. Hence there will be leaf nodes (nodes without any outedges) in the graph. These represent program units which do not invoke any subprograms. Hence the flow graphs for these program units are known to be completely labelled. DAVE now continues by carefully analyzing these program units' flow graphs.

Once a program unit's flow graph is completely labelled it is possible to determine the pattern of references to and definitions of each of the program unit's variables for each of the program unit's execution sequences. Uninitialized and dead variables are found by examining these patterns. In DAVE a variable's pattern of references and definitions is determined and examined by a depth-first search procedure (described in detail in Reference 18) which executes in time proportional to the number of edges in the flow graph. The search procedure is repeated for each variable in the program unit. It classifies each variable as either non-input, input or strict input and either non-output, output or strict output. A variable is classified input if along some, but not all, execution sequences through the program unit the variable is referenced before it is defined. If there is no such execution sequence, the variable is classified non-input. If the variable is referenced before definition along all execution sequences, the variable is classified strict input. Similarly, the variable is classified output if along some, but not all, execution sequences in the program unit the variable is defined. If there is no such execution sequence the variable is classified non-output. If it is defined along all execution sequences, it is classified strict output.

These classifications having been made, DAVE begins its search for anomalies. If a local variable is classified strict input, it is clear that a type 1 anomaly will occur, and an error message is produced. If a local variable is classified input, then a type 1 anomaly exists for some, but not all, sequences of statements. In recognition of the fact that these sequences may not actually be executable in response to any input data, DAVE produces a warning message describing the possibility of executing an anomaly bearing sequence of statements. DAVE performs similar scans for type 2 anomalies by executing searches from a definition of a local variable to determine whether the subprogram terminates or redefines the variable before referencing it.

The determination of the input/output status of non-local variables (i.e., parameters and COMMON variables) of leaf subprograms is not used immediately in the detection of anomalies, but rather is used to enable DAVE's analysis to continue for higher level program units. The program call graph is used to locate all invocations of the leaf subprograms, and now the nodes corresponding to these invocations are labelled. At the end of this process, some non-leaf subprograms have become completely labelled and the depth-first search procedure can be applied to them. This process continues until eventually the main program itself is searched.

The process of using the input/output classification of a non-local variable of an invoked subprogram to label an invoking node is worthy of some elaboration here as it incorporates a number of useful error checks. DAVE first compares argument and parameter lists for agreement in length and type. Lack of agreement is considered an error. Next, parameter output classifications are compared to arguments. If a parameter is classified as output or strict output and the corresponding argument is a constant, expression or subprogram name, DAVE produces a message. COMMON variables which carry data into or out of the invoked subprogram are identified at this time and messages describing them are made available for use as documentation. Finally, the variables in the invoking statement are examined to see whether any of them is used both as an input and an output in separate subprogram invocations. If so, DAVE has detected an illegal side effect, and produces a message identifying it.

After DAVE has searched the main program, it examines the input classifications of its COMMON variables. Error or warning messages are generated for each COMMON variable which is typed strict input or input but is not initialized in a BLOCK DATA subprogram.

IMPLEMENTATION DATA

DAVE is implemented as a Fortran program consisting of approximately 25,000 source statements. It operates in four overlaid phases, the largest of which occupies 50,000 decimal words of central memory on the CDC 6400. DAVE is written almost entirely in machine independent ANSI Standard Fortran. Some non-Standard and machine dependent coding practices seemed expedient, however, and they are quarantined to a small number of small subprograms. DAVE was developed on the CDC 6400 at the University of Colorado, but has been successfully moved to a CDC 7600 and two machines in the IBM 360/370 series. Installations on a Univac 1100 series machine and a Honeywell 6000 series machine are planned for the near future.

Under its current configuration DAVE is able to process a program consisting of a few dozen subprograms, each of which may contain no more than 200-250 source statements. These limits depend entirely upon internal table and scratch array sizes, and have been quickly altered to produce different experimental configurations. At this writing, the largest body of code which DAVE has processed has been a 2700 source statement subprogram library. DAVE is currently operational, however, on a machine with sufficient central memory to enable it to process its own source code, and this should be accomplished in the near future. The analysis of a source program by DAVE has been observed to require an average of 0.3 seconds of central processor time per source statement on the CDC 6400 and to cost approximately six to eight cents per source statement under the University of Colorado Computing Center charge algorithm.

SUMMARY OF EXPERIENCE TO DATE

DAVE has been operational on an experimental basis for a few months to date. During this time we have seen evidence that it can be a valuable tool in the production of high quality, error-free programs. Most of our experience has come from using DAVE in validating completed programs. These included a highly respected matrix manipulation system, several recent algorithms taken from the *ACM Transactions on Mathematical Software,* and a program submitted as a part of a Master's Thesis in Computer Science. Errors were detected in some of the algorithms, and the Master's Thesis was found to have numerous errors. In most cases the errors were of the type that would hamper program portability, such as reference to uninitialized variables which should have been initialized to zero, reference to exhausted DO loop indices and subprogram invocations with mismatched argument and parameter lists. In each case, the errors did not seem to prevent successful execution on the author's computer, but seemed likely to cause trouble if executed elsewhere. (There was some suspicion, however, that some of the erroneous subprogram invocations were imbedded in program segments which had never been tested or were unexecutable.)

Perhaps the most surprising observation was that DAVE's messages often gave unexpected insight into the author's coding style. For example, a program for which DAVE produced numerous type 2 anomaly warning messages did not prove to be incorrect, but rather it contained numerous loops in which indices and counters were updated immediately before DO loop endings. It was discovered that the author tended to favor WHILE loop constructions which are often awkward in Fortran. This was observed by DAVE. As another example, some programs contained subroutine definitions which did not use some parameters either as input or output. This was observed to be a symptom of the fact that the code had evolved, but not been carefully polished. We acknowledge that in the first case the author should probably have coded in a more comfortable language, and in the second the program was probably not thoroughly designed before coding began. DAVE can do nothing to prevent these serious breaches of good programming practice. It was surprising and gratifying, however, to discover that DAVE could often strongly indicate their presence—a capability which we believe is quite useful.

We have had less experience in using DAVE as an error detection aid during program development. This seems paradoxical because we feel that DAVE is very well suited to aiding the detection of subtle errors, thereby speeding program development. The high cost of using and the awkwardness in accessing the current version of DAVE, however, forestalled its use in many cases. DAVE's accessing procedures have recently been streamlined, but the high cost of using the system is attributable to a decision made during development of the prototype system to favor flexibility over efficiency. Hence high costs are likely to remain for the foreseeable future. As a consequence of this, the few programs which DAVE helped to debug all had subtle errors which had defied earlier concerted efforts at detection. DAVE was usually able to point rather directly at these. Such errors as camouflaged misspellings (e.g., CARD instead of CARDS) and mismatched argument/parameter lists were discovered in this way.

Our experience has not been entirely positive. An obvious and troublesome difficulty is DAVE's copious output. As already illustrated, a single error often generates numerous messages. Moreover, we have observed that some messages are rarely symptoms of errors. The net effect is that human analysts are often reluctant to pursue all of DAVE's messages, thereby raising the possibility that errors whose symptoms have been detected will go unnoticed. DAVE users have also complained about the unclear wording of many messages. All of these human interface problems must be solved lest DAVE's useful capabilities be buried under an avalanche of opaque verbiage.

PROBLEM AREAS AND FUTURE WORK

We consider the current DAVE system to be a working prototype. Consequently, as might be expected, it has neither the speed nor complete processing capabilities which might be expected of a polished system.

The purpose of this section is to describe the areas in which we feel DAVE is deficient and to indicate where and how improvements might be made.

One of the most immediate problems is that DAVE was designed to analyze only programs written in ANSI Standard Fortran. DAVE has since been liberalized to accept most of the Fortran dialects available on CDC equipment. Little effort, however, has been devoted to the problems of accepting other dialects. Many of the changes required in order to accept such dialects appear to be straightforward, but it is worthwhile to note that some features of some dialects (e.g., the ENTRY feature found in FORTRAN V[19]) cannot be properly analyzed by DAVE without substantial alterations.

More serious is the fact that there are a number of features of ANSI Standard Fortran which are currently incorrectly or inadequately handled by DAVE. A notable and discouraging example of this is the treatment of arrays. Currently DAVE treats all arrays as simple variables, thereby blurring all distinctions between array elements and eliminating the possibility of detecting certain anomalous uses of the individual elements. Unfortunately, there are fundamental theoretical reasons why patterns of array references in an arbitrary program can never be completely analyzed by a static analysis system such as DAVE.

As already noted, the call graph of a Fortran program may not be acyclic even though the program is incapable of ever executing a recursive calling chain. Such a program cannot be analyzed by DAVE. The most promising solution to this problem seems to be to adapt DAVE so that it is able to analyze recursive programs. This is an interesting and worthwhile problem which seems solvable, and would move DAVE in the direction of being able to analyze programs written in other languages such as ALGOL and PL/I.

DAVE is also currently unable to build the complete call graph for programs in which subprogram names are passed as parameters. Hence DAVE cannot analyze such programs. As algorithm due to Kallal and Osterweil[20] is capable of building the call graph of such a program. This algorithm will probably be incorporated into future versions of DAVE.

Other problems are encountered by DAVE in trying to analyze programs containing extensive or tricky uses of aliasing constructs such as EQUIVALENCE statements and restructured COMMON lists. Most of these will be overcome in future versions of DAVE by using well-known compiling techniques. Others, such as using two EQUIVALENCE'd variables as arguments to the same subprogram, challenge some of DAVE's basic assumptions, and may never be satisfactorally solved.

Programs in which variables become undefined (e.g., the exhaustion of a DO loop causes the undefinition of the DO index) may, under certain unusual circumstances, be incorrectly analyzed. This results from our tardy recognition that variables must be typed with respect to undefinition just as they are typed with respect to input and output (i.e., they must be typed as non-undefined, undefined, or strict undefined). We have developed algorithms for performing and correctly employing this typing of undefinition, but have not yet incorporated them into DAVE.

Finally, we are actively exploring the relationship between static testing aids and global program optimization. Our investigation[21] has shown that existing algorithms in global optimization can readily be harnessed to do much of the analysis done by DAVE. Hence we foresee the incorporation of systems such as DAVE into a future generation of compilers.

REFERENCES

1. Dijkstra, E. W., "Notes on Structured Programming," in *Structured Programming*, by O. J. Dahl, E. W. Dijkstra and C. A. R. Hoare, Academic Press, London and New York, 1972.
2. Wirth, N., "Program Development by Stepwise Refinement," *CACM* 14, pp. 221-227, April 1974.
3. Mills, H. D., "Top-Down Programming in Large Systems," in *Debugging Techniques in Large Systems*, R. Rustin (ed.), Prentice-Hall, Englewood Cliffs, N.J., 1971, pp. 41-45.
4. Wirth, N., "An Assessment of the Programming Language PASCAL," *IEEE Transactions on Software Engineering*, SE-1, pp. 192-198, June 1975.
5. Gannon, J. D. and J. J. Horning, "Language Design for Program Reliability," *IEEE Transactions on Software Engineering*, SE-1, pp. 179-191, June 1975.
6. Mills, H. D., "How to Write Correct Programs and Know It," *Proceedings of the 1975 International Conference on Reliable Software*, IEEE Cat. No. 75CH0940-7CSR, pp. 363-370.
7. Baker, F. T., "Chief Programmer Team Management of Production Programming," *IBM Systems Journal*, 11, pp. 56-73, 1972.
8. Good, D. I., R. L. London and W. W. Bledsoe, "An Interactive Program Verification System," *IEEE Transactions on Software Engineering*, SE-1, pp. 59-67, March 1975.
9. Balzer, R. M., "EXDAMS: Extendable Debugging and Monitoring System," *AFIPS 1969 SJCC*, 34 AFIPS Press, Montvale, N.J., pp. 567-580.
10. Fairley, R. E., "An Experimental Program Testing Facility," *Proceedings of the First National Conference on Software Engineering*, IEEE Cat. No. 75CH0992-8C, pp. 47-52.
11. Stucki, L. G., "Automatic Generation of Self-Metric Software," *Proceedings of the 1973 IEEE Symposium on Computer Software Reliability*, IEEE Cat. No. 73C40741-9CSR, pp. 94-100.
12. Ramamoorthy, C. V. and S-B. F. Ho, "Testing Large Software with Automated Software Evaluation Systems," *IEEE Transactions on Software Engineering*, SE-1, pp. 46-58, March 1975.
13. Miller, E. F., Jr. and R. A. Melton, "Automated Generation of Testcase Datasets," *Proceedings of the 1975 International Conference on Reliable Software*, IEEE Cat. No. 75CH0940-7CSR, pp. 51-58.
14. Krause, K. A., R. W. Smith and M. A. Goodwin, "Optimal Software Test Planning Through Automated Network Analysis," *Proceedings of the 1973 IEEE Symposium on Computer Software Reliability*, IEEE Cat. No. 73C40741-9CSR, pp. 18-22.
15. American National Standards Institute, *FORTRAN*, ANSI X3.9, 1966.

16. Schaefer, M., *A Mathematical Theory of Global Program Optimization*, Prentice-Hall, Englewood Cliffs, N.J., 1973.

17. Osterweil, L. J. and L. D. Fosdick, *Data Flow Analysis as an Aid in Documentation, Assertion Generation, Validation and Error Detection,* University of Colorado Department of Computer Science Technical Report No. CU-CS-055-74.

18. Osterweil, L. J. and L. D. Fosdick, "DAVE—A Validation, Error Detection, and Documentation System for Fortran Programs," *Software Practice & Experience* (to appear).

19. Univac Division of Sperry Rand Corporation, *UNIVAC 1108 FORTRAN V,* Cat. No. UP-4060 Rev. 1, Sperry Rand Corporation, pp. 4-18 to 4-19.

20. Kallal, V. and L. J. Osterweil, (to appear as University of Colorado Department of Computer Science Technical Report).

21. Fosdick, L. D. and L. J. Osterweil, "Validation and Global Optimization of Programs," *Proceedings of the Fourth Texas Conference on Computing Systems,* 1975.

A Dynamic (FORTRAN) programming system

by JULIUS A. ARCHIBALD, JR.
State University College
Plattsburgh, New York

ABSTRACT

In recent years, new insights into the nature of programming languages have been obtained from the comparative study of natural and programming languages. These studies reveal that programming languages are deficient in their ability to adapt both to new requirements and new means for communicating thoughts. A means of alleviating these difficulties, through a dynamic, structured expansion of an established programming language (FORTRAN) is provided.

INTRODUCTION

During recent years, there have been some rather significant advancements in man's understanding of programming languages. Some ten or so years ago, and continuing into the present, we find a continuing growth of interest in the concepts of structured programming.[1,2] More recently, there has been the suggestion that the nature of programming languages can be better understood by the study of natural languages and the drawing of analogies between these two different types of languages.[3] One particularly useful analogy has been suggested, that between English, a poor, but useful natural language, and FORTRAN, a poor, but useful, programming language.[4] In doing this, we note that one of the major differences between these media of thought is their adaptability to changing requirements. Specifically, we note, both through a study of literature, and a review of our own usage of the language, that English has readily adapted both to the needs of expressing new thoughts, and to the needs of better ways of expressing and communicating old thoughts. English has thus evolved in a timely manner. Indeed, the development of English has been concurrent with, rather than lagging behind, the development of human knowledge. We also note, regretfully, that programming languages in general, and FORTRAN in particular do not share this characteristic. Programming languages as we now know them, are incapable of dynamic development or evolvement. Programming methods have changed, and languages have failed to keep pace.

The static nature of programming languages is a problem of particular concern today. The earliest programming languages, such as FORTRAN, were, of necessity, created before our present day understandings of the nature and structure of algorithmic processes and programming languages were attained. (Versions of FORTRAN were in use some ten years before the notion of structure was developed.) The time has come for us to benefit from the new understandings of the structure of algorithmic processes and programming languages developed in the last ten years within the framework of FORTRAN. The ideal solution would be to extend FORTRAN so as to include the new structures.

There are two problems. First of all, FORTRAN has been standardized.[5,6] On the positive side, this was good for purposes of definition. We now have a precise understanding as to exactly what is FORTRAN and what is not FORTRAN. Even more important, this understanding is the same in Boston, Atlanta, Los Angeles, and Seattle. The definition is not subject to local dialects. On the negative side, however, along with standardization came stagnation. The existence of a standard has successfully stifled the initiative needed to make the language flexible, and adaptive. To further complicate matters, those who were not content with the status quo mandated by the standard have ventured off into their own private extensions in such a manner as to produce a set of mutually incompatible super-languages of the original common language. The overall effect has not been beneficial to the development of programming languages.

The second problem is, in reality, not a problem of the programming languages themselves, but rather a problem resulting from differences in the use of natural and programming languages. The constructs of English become meaningful as a result of interpretation by thought processes resident in the human brain. The constructs of FORTRAN become meaningful as a result of interpretation by compilers (or interpreters) resident in a computer's memory. Thus, the increased flexibility (or adaptability) of natural languages over programming languages is not a question of the relative merits of the languages themselves, but rather a matter of the superior ability of the human brain, as compared to a compiler, to program itself (or be programmed) to interpret new constructs. Thus, in order to attain for a programming language the flexibility and adaptability of a natural language, we must consider not only the language itself, but also the compiler (at least in general terms), as they are known to exist for the language. We thus must act upon Wirth's conclusion that "Language design is compiler construction."[7]

We refer here to a programming system on a given machine as being made up of a specific language and the compiler or interpreter used to interpret that language on the machine of concern. We will also refer to FORTRAN programming systems on a machine independent basis in the same manner that FORTRAN as a language is regarded on a machine independent basis. In the remainder of this paper, we will be working on obtaining flexibility and adaptability not merely for the language itself, but rather for the overall programming system. The objective is a dynamic, flexible extension of FORTRAN, as previously defined.[5] It will be accomplished through modification of the FORTRAN programming system.

It is noted in passing that the present use of static structured extensions of FORTRAN, which must be converted to standard FORTRAN, through the use of preprocessors, falls short of the above stated objective. They provide structure without flexibility. The need for a dynamic language has already been noted in the literature.[8] Conventional pre-processors simply are not dynamic. A second problem is that the preprocessors are not always machine independent. The various extensions themselves are not all consistent.

The approach of limiting oneself to a subset of the existing, standard FORTRAN from which things resembling structures can be formed is also inappropriate. This approach involves the further limitation of an already inadequate medium, rather than an extension of the medium to meet new challenges. Said in other words, this approach also fails to provide flexibility.

There is one more remaining question: "Why FORTRAN?" Again, the analogy with English. Users of English (both native and adopted) love to sit around and complain about how poor a language it is, how bad the grammar is, and how impossible spelling is. No one, however, has taken any serious steps to eliminate English. The best effort, to date, was the invention of a contrived artificial language, Esperanto,[3] a language with lots of merit and no potential. The problem is that too many people have already done, and are continuing to be doing, too many things with English to make a change feasible. Consider, for a moment, the problems that some of us are already experiencing from the change of just one small part of the language, namely the system of measures, to metric units. The same is true of FORTRAN, too many people have already done, and are continuing to do, too many things in FORTRAN to make a change feasible. FORTRAN is where the action is. The language, PL/I, intended to be a partial remedy, has no more potential than Esperanto. We are, as a matter of fact, suffering from addiction to FORTRAN. In the present situation, the pain of continued use is less than the pain of withdrawal. We thus support previous conclusions of others[3] as well as ourselves.[9]

IMPLEMENTATION

It has been shown that all programs, regardless of the language in which they are written, can be composed of three fundamental structures.[1] These structures consist of a sequential or composite structure, some variant of a predicate structure, and some variant of a repetitive structure. Regardless of the choice of variant, the result, that these structures are sufficient, remains valid.[10] (We do include, in our implementation, a fourth structure, the case structure, fully realizing that its use is not essential.) Thus, as will be seen, we will limit ourselves to D, D′, and BJn structures.[11] The point is that each structure is made up of certain, fixed parts, which, because they are independent of the language concerned, are referred to by psycholinguists as being "linguistic universals." [12]

The method of implementation that we use is to divide each program into two divisions. The first division consists of a definition of the form to be used for each part of each of the fundamental structures (or variants of the structures) to be used within the program. (We note that there may be several different definitions for each of the fundamental structures within the programs.) These structure definitions are followed by the second division, consisting of the source program itself, written in standard FORTRAN augmented by the just defined structures. Each programmer is free to define the fundamental structure in any manner that suits his (or her) convenience, thereby providing flexibility. Division one of the compilation will be the conversion of the user defined structures within the source program into standard FORTRAN statements, and the insertion of the resulting statements into appropriate places of the original source. In most situations, we "very strongly convert" in the sense of Ledgard and Marcotty, the various D, D′, and BJn structures to standard FORTRAN structures.[11] This conversion is followed by a routine FORTRAN compilation of the entire converted program. Any construct not part of standard FORTRAN must have been defined and converted. A program without structure definitions is assumed to be in standard FORTRAN, and, as such, does not require a division one of the compilation.

This differs from pre-processing, as it already exists today, in that each programmer, on a dynamic basis, creates his own form or forms for the fundamental structures. We reiterate that several different forms of each structure may be used in each program. There are no pre-established forms for the structures themselves, only the parts of the structures remain invariant. This is the feature that provides the flexibility absent in the more conventional pre-processing.

All terminology used in describing structures will be compatible with FORTRAN usage. Thus, if in defining a predicate structure, the condition itself is referred to as being a logical expression, then the condition will require no further definition. Rather, it shall be assumed that all of the attributes and characteristics of FORTRAN logical expressions will apply.

This compatibility with FORTRAN will also apply to the structure definition statements, e.g., such statements will begin in Column 7, etc.

As a specific of the implementation, the actual conversion of the user defined structures into FORTRAN must itself be done in FORTRAN, and the resulting conversion must result in standard FORTRAN statements.[5] (This in itself is difficult because of the incomplete manner in which strings are defined in FORTRAN.) The conversion of user defined structures will require the insertion of new statements labels, and may, as well, require new variable names. In those cases where the existing FORTRAN can readily be adapted to accept variable names and statement labels beginning with previously illegal characters, such as $, or #, the conversion will be simplified. In some other cases, the implementation may permit the programmer to designate and reserve specific sets of labels and names for the conversion process. Neither of these, however, can be a limitation, and, in the general case, the source program must be completely pre-scanned so that blocks of legal, unused statement labels and variable names may be identified for use in the structure conversion.

As a part of the structure conversion phase, the original structure statements will be converted into comments, so that they can be retained for documentary purposes.

STRUCTURE DEFINITION

It is emphasized that, on a program by program basis, the programmer is free to define any number of structures (including none at all) that suits his (or her) convenience in the writing of the program. The programmer will be free to define his own structures, so long as he (or she) retains all of the essential parts for each structure. Thus, in what follows, the method of definition will be presented. Examples will be included for illustrative purposes only.

While no attempt is made to either prescribe or limit the forms of specific structures, certainly, all of the control structures occurring in the more common languages (e.g., ALGOL), expressed in any natural language using the Latin alphabet, should be definable for use in the dynamic FORTRAN programming system. In our dynamic system, words like "si", "alors", and "autre" could easily be used in place of "if", "then", and "else".

As indicated previously, the actual form of the structure definition depends upon the structure itself. The structures to be used are defined in a "structure division", placed in front of the main program. The first statement of each definition (beginning in Column 7) will be one of the following depending upon the structure:

```
STRUCTURE SEQUENTIAL
STRUCTURE PREDICATE
STRUCTURE REPETITIVE
STRUCTURE CASE
```

Each of these structures has its own, unique parts, which must be defined. Starting in Column 7, the component of the structure is indicated. That is followed by the statement to be used in the program to define the structure.

As an overall program organization, each program consists of a large sequential structure (which need not be explicitly defined). Within this structure, predicate, repetitive, and case structures, if needed, must be programmed in a form defined either in the structure section, or in standard FORTRAN. Structures may be nested within other structures. Indication of sequential structures is optional, except that they must be indicated explicitly when they are made up of more than one statement and are contained within predicate, case, or repetitive structures. At the other extreme, each statement is, by default, a sequential structure of one statement. There is no need to ever explicitly define a sequential structure of only one statement, even when it is inside of a predicate, repetitive, or case structure.

DEFINITION OF SEQUENTIAL STRUCTURES

The sequential structure is extremely trivial. To define a sequential structure or a sequential block, it is merely necessary to indicate its opening and its closing form. For example, a programmer might define the structure as follows:

```
STRUCTURE SEQUENTIAL
OPENING BEGIN
CLOSING END
```

This will cause the programming system to recognize groups of statements and/or structures between the defined OPENING and CLOSING statement brackets as a sequential structure. Each individual statement will be treated as a sequential structure without being so defined, and without having statement brackets. In the general case the defined OPENING statement will be converted into a comment, and otherwise ignored. The defined CLOSING statement will be converted into a comment, and a CONTINUE with a legal label will be inserted just ahead of the converted END. The following special cases are recognized:

1. If the entire program is included as a sequential block, the defined CLOSING statement is converted into a conventional FORTRAN END statement. (It is anticipated, however, that nonexecutable statements, such as DIMENSION and FORMAT, will be placed outside of such blocks.)
2. If the structure is the THEN or ELSE part of a predicate structure, a PROCEDURE part of a replicative structure, or an ALTERNATIVE part of a case structure, a CONTINUE with a legal label is inserted just after the converted OPENING statement.
3. If the defined OPENING statement is labeled, a CONTINUE statement with this label is inserted just after the defined OPENING statement.
4. If the defined CLOSING statement is labeled, this label will be used on the CONTINUE inserted just before it.

DEFINITION OF PREDICATE STRUCTURES

The predicate structure consists of from two to four parts: a condition, an affirmative or condition—true alternative, an optional negative or condition—false alternative, and an optional closing. The condition itself functions as the opening of the structure. The two alternatives will be identified as sequential structures (possibly containing other structures). A closing may optionally be defined for the structure. If, however, no such closing is defined, the structure will be assumed to terminate at the end of the negative alternative, if any, or at the end of the positive alternative if there is no negative alternative.

As an example, consider:

```
STRUCTURE PREDICATE
OPENING IF (logical expression)
AFFIRMATIVE THEN (structure)
NEGATIVE ELSE (structure)
CLOSING END IF
```

(The use of parentheses is for generic purposes, i.e., any logical expression legal within FORTRAN, or any structure defined for the current program, may be used.)

The opening statement will be converted to a comment. A standard logical IF for the negative of the logical expression in the opening statement will be created to transfer to the negative alternative (or the closing if there is no negative alternative). This statement will carry the same label as the original opening in the structured form, if any. The positive alternative (a structure) will follow, in line, terminating with a converter generated GO TO the closing of the predicate structure. The negative alternative will then follow in line. A labeled CONTINUE will be inserted as the final statement (or closing) of the predicate structure. The user supplied indicators of the alternatives and the closing will be converted into comments.

It is noted that the requirement for a condition and an affirmative alternative structure precludes the interpretation of a standard FORTRAN logical IF as a defined structure. Use of the standard logical IF, followed by a single statement "then procedure" (other than the GO TO) is encouraged in the dynamic FORTRAN programming system. Such logical IF's possess all of the virtues of structure. Moreover, as part of standard FORTRAN, they require no further definition.

DEFINITION OF REPETITIVE STRUCTURES

The repetitive structure is, perhaps, the most difficult to define. This is a natural result of the fact that the repetitive structure permits a large number of variations. A controlled loop, whether or not it is arranged as a definite structure, has certain recognizable parts. These always include a body or procedure that is repeated many times, and a test that is performed many times, to determine whether or not to leave the loop. In some cases, the loop is controlled by a counter or index that must be initialized once and incremented many times. In some cases, there are data values to be initialized once. The testing may be done before each performance of the procedure, after each performance of the procedure or at a specific point within the procedure. In the latter case, there are, in effect, two procedures separated by a test, and organized in such a manner that there will be many performances of the two procedures, in order. (In what follows, we will not be limited as to the number of possible procedures to be repeated. We have implemented the full Omega-K structure of Bohm and Jacopini.[1]) We note, further, that there are two possible tests for leaving a loop: the loop may be continued UNTIL a certain condition becomes true, or it may be continued WHILE a certain condition remains true. (These are reverses of each other. In the former case, repeat on condition false, in the second repeat on condition true. Negating the condition permits switching of the test.)

Thus, in describing a repetitive structure, there is a mandatory opening section, an optional initialization, one or more procedures to be repeated, one or more tests for continuation or completion of the loop either before the first procedure, after the last procedure, or between any two procedures and finally a structure closing.

As an example consider:

```
STRUCTURE REPETITIVE
OPENING PERFORM
INITIAL ESTABLISH (structure) *
```

* The INITIAL part is optional.

```
CONTINUE or
COMPLETE      TEST (logical expression) ‡
PROCEDURE (structure)**
CONTINUE or
COMPLETE      TEST (logical expression) ‡
PROCEDURE (structure)**
CONTINUE or
COMPLETE      TEST (logical expression) ‡
   .      .
   .      .
   .      .
CLOSING END PERFORM
```

During the structure conversion phase of the program, the opening statement will be converted to a comment, and the initialization part expanded as previously described. A labeled CONTINUE statement will be inserted either immediately after the initialization, if any, or after the opening. Each continue part or complete part will be preceded and followed by labeled CONTINUE statements, to be inserted if not provided by the programmer. (Two of these parts coming together, however, will not have a CONTINUE inserted between them.) The continue and complete parts, as logical expressions, will be incorporated into logical IF statements, as follows:

```
CONTINUE part
    IF (.NOT. expression) GO TO closing
COMPLETE part
    IF (expression) GO TO closing
```

The various procedure parts will be converted as previously described. Immediately before the closing component, a GO TO the beginning of the first procedure will be inserted by the converter. The closing itself will be converted into a labeled CONTINUE. As in the case of all structures, all statements requiring conversion will be converted into comments.

DEFINITION OF CASE STRUCTURES

The case structure involves the identification of an (integer-valued) arithmetic expression which functions as an index and a set of alternatives or choices to be executed depending upon the value of the index, i.e., if the index is 1, choice 1 only will be executed, etc. For purposes of implementation, the arithmetic expression functions as the opening of the structure. A closing part is mandatory.

As an example, consider:

```
STRUCTURE CASE
OPENING EXAMINE (arithmetic expression)
ALTERNATIVE n CASEn (structure)*
.
.
.
CLOSING END CASE
```

The opening statement itself is converted to a comment. If the arithmetic expression is not an integer variable name, an arithmetic assignment statement will be generated, setting the expression to an integer variable. The IFIX function will be inserted if needed. A computed GO TO statement will then be created, to send control to an appropriate alternative. The arithmetic assignment statement, if any, or the computed GO TO if no assignment statement is needed, will carry the same label as the original opening in the structured form, if any. Each of the alternatives will end with a GO TO the closing statement, which, itself, will be converted to a CONTINUE with a system generated label.

GENERAL PROVISIONS

During the process of interpreting the user supplied structure definition, the definition statements will be converted into comments. A blank comment statement will be inserted following every closing component.

All comments included in the original program will be retained during structure conversion.

It is generally recommended that structures be indented to facilitate their recognition. In using this dynamic programming system, no indentation rules are imposed upon the user. Indentation supplied by the user (if any) will be retained by the converter.

SAMPLES

In what follows, learning exercises will be shown written in a form for the FORTRAN programming system, and then converted into standard FORTRAN. They are intended purely as a demonstration of the concepts previously described.

Sample 1.—Table of roots of integer valued real numbers.

A. Source in Dynamic FORTRAN

```
STRUCTURE SEQUENTIAL
OPENING START
CLOSING FINISH
STRUCTURE REPETITIVE
OPENING REPEAT
INITIAL SET
PROCEDURE
```

** One PROCEDURE part required, the remainder are optional.
‡ One CONTINUE or COMPLETE part is required, the remainder are optional. A repetitive structure may be defined with both a continue and a complete part.

* Normally, there will be three or more of these components. The lower case n is used to indicate an integer number: 1,2, . . . maximum value.

Sample 1 continued

A. Source in Dynamic FORTRAN continued

```
      COMPLETE TEST
      CLOSING END REPEAT
      STRUCTURE REPETITIVE
      OPENING ITERATE
      INITIAL SET
      PROCEDURE
      COMPLETE CONVERGE
      CLOSING END ITERATE
      DIMENSION Y(10)
      START
      WRITE (6,1001) (N,N=1,10)
      REPEAT
        SET
        I=1
        START
        X=FLOAT(I)
        Y(1)=X
        REPEAT
          SET
          J=2
          START
          ITERATE
            SET
            START
            V=FLOAT(J)
            Z3=1.
            FINISH
            START
            Z=Z3
            Z1=(V-1.)*Z
            Z2=X/Z**(J-1)
            Z3=(Z1+Z2)/V
            FINISH
            CONVERGE
            ABS(Z-Z3) .LT. X*1.E-6
            END ITERATE
          Y(J)=Z3
          J=J+1
          FINISH
          TEST
          J .GT. 10
          END REPEAT
            WRITE (6,1002) Y
            I=I+1
            FINISH
            TEST
            I .GT. 50
            END REPEAT
          STOP
          FINISH
     1001 FORMAT (1H1,10(3X,5HROOT ,I2))
     1002 FORMAT (1H ,10F10.6)
          END
```

B. Converted Standard FORTRAN

```
C       STRUCTURE SEQUENTIAL
C       OPENING START
C       CLOSING FINISH
C
C       STRUCTURE REPETITIVE
C       OPENING REPEAT
C       INITIAL SET
C       PROCEDURE
C       COMPLETE TEST
C       CLOSING END REPEAT
C
C       STRUCTURE REPETITIVE
C       OPENING ITERATE
C       INITIAL SET
C       PROCEDURE
C       COMPLETE CONVERGE
C       CLOSING END ITERATE
C
        DIMENSION Y(10)
C       START
        WRITE (6,1001) (N, N=1,10)
C       REPEAT
C         SET
          I=1
90001     CONTINUE
C         START
          X=FLOAT(I)
          Y(1)=X
C         REPEAT
C           SET
            J=2
90002       CONTINUE
C           START
C           ITERATE
C             SET
C             START
              V=FLOAT(J)
              Z3=1.
90003         CONTINUE
C             FINISH
90004         CONTINUE
C             START
              Z=Z3
              Z1=(V-1.)*Z
              Z2=X/Z**(J-1)
              Z3=(Z1+Z2)/V
90005         CONTINUE
C               FINISH
C             CONVERGE
              IF (ABS(Z-Z3) .LT. X*1.E-6) GO
                TO 90006
              GO TO 90004
90006         CONTINUE
C             END ITERATE
            Y(J)=Z3
            J=J+1
90007       CONTINUE
C           FINISH
```

Sample 1 continued

B. Converted Standard FORTRAN continued

```
C            TEST
             IF (J.GT. 10) GO TO 90008
             GO TO 90002
90008        CONTINUE
C            END REPEAT
          WRITE (6,1002) Y
          I=I+1
90009     CONTINUE
C         FINISH
C         TEST
          IF (I .GT. 50) GO TO 90010
          GO TO 90001
90010     CONTINUE
C         END REPEAT
       STOP
90011  CONTINUE
C      FINISH
 1001  FORMAT (1H1, 10(3X,5HROOT ,I2))
 1002  FORMAT (1H ,10F10.6)
       END
```

Sample 2.—Replacement sort of forty random numbers

A. Source in Dynamic FORTRAN

```
   STRUCTURE SEQUENTIAL
   OPENING BEGIN
   CLOSING END
   STRUCTURE PREDICATE
   OPENING TEST
   AFFIRMATIVE THEN
   CLOSING END TEST
   STRUCTURE REPETITIVE
   OPENING LOOP
   INITIAL SET
   PROCEDURE
   CONTINUE WHILE
   CLOSING END LOOP
   DIMENSION D(40)
   BEGIN
   WRITE (6,91)
   LOOP
     SET
     I=1
     BEGIN
     CALL BGHT(D(I))
     WRITE (6,92) D(I)
     I=I+1
     END
     WHILE
     I .LE. 40
     END LOOP
   WRITE (6,93)
   LOOP
     SET
     I=1
     BEGIN
       LOOP
         SET
         BEGIN
         IMIN=I
         J=I+1
         END
         BEGIN
         TEST
           D(J) .LT. D(IMIN)
           THEN
           IMIN=J
           END TEST
         J=J+1
         END
         WHILE
         J .LE. 40
         END LOOP
       T=D(I)
       D(I) =D(IMIN)
       D(IMIN) =T
       I=I+1
       END
       WHILE
       I .LE. 39
       END LOOP
   LOOP
     SET
     I=1
     BEGIN
     WRITE (6,92) D(I)
     I=I+1
     END
     WHILE
     I .LE. 40
     END LOOP
   STOP
   END
91 FORMAT (1H1, 38HSORT OF FORTY RANDOM
       NUMBERS, UNSORTED, //)
92 FORMAT (1H ,E15.8)
93 FORMAT (1H1, 36HSORT OF FORTY RANDOM
       NUMBERS, SORTED, //)
   END
```

B. Converted Standard FORTRAN

```
C      STRUCTURE SEQUENTIAL
C      OPENING BEGIN
C      CLOSING END
C
C      STRUCTURE PREDICATE
```

Sample 2 continued

B. Converted Standard FORTRAN continued

```
C       OPENING TEST
C       AFFIRMATIVE THEN
C       CLOSING END TEST
C
C       STRUCTURE REPETITIVE
C       OPENING LOOP
C       PROCEDURE
C       CONTINUE WHILE
C       CLOSING END LOOP
C
        DIMENSION D(40)
C       BEGIN
          WRITE (6,91)
C         LOOP
C           SET
            I=1
90001       CONTINUE
C           BEGIN
              CALL BGHT(D(I))
              WRITE (6,92) D(I)
              I=I+1
90002         CONTINUE
C             END
C           WHILE
            IF (.NOT. (I .LE. 40)) GO TO 90003
            GO TO 90001
90003       CONTINUE
C           END LOOP
          WRITE (6,93)
C         LOOP
C           SET
            I=1
90004     CONTINUE
C         BEGIN
C           LOOP
C             SET
C             BEGIN
              IMIN=I
              J=I+1
90005         CONTINUE
C             END
90006       CONTINUE
C           BEGIN
C             TEST
              IF (.NOT. (D(J) .LT. D (IMIN) ))
                  GO TO 90007
C             THEN
              IMIN=J
90007         CONTINUE
C             END TEST
            J=J+1
90008       CONTINUE
C           END
C           WHILE
            IF (.NOT. (J .LE. 40)) GO TO 90009
            GO TO 90006
90009       CONTINUE
C           END LOOP
          T=D(I)
          D(I)=D(IMIN)
          D(IMIN)=T
          I=I+1
90010     CONTINUE
C         END
C         WHILE
          IF (.NOT. (I .LE. 39)) GO TO 90011
          GO TO 90004
90011     CONTINUE
          END LOOP
C         LOOP
C           SET
            I=1
90012       CONTINUE
C           BEGIN
              WRITE (6,92) D(I)
              I=I+1
90013         CONTINUE
C             END
C           WHILE
            IF (.NOT. (I .LE. 40)) GO TO 90014
            GO TO 90012
90014       CONTINUE
C           END LOOP
          STOP
90015     CONTINUE
C         END
91        FORMAT (1H1, "SORT OF FORTY RAN-
             DOM NUMBERS, UNSORTED" //)
92        FORMAT (1H ,F15.8)
93        FORMAT (1H1, "SORT OF FORTY RAN-
             DOM NUMBERS, SORTED" //)
          END
```

CONCLUSIONS

The foregoing discussion and samples demonstrate a method for moving the practice of programming into the nineteen seventies without abandoning FORTRAN. The following observations are made:

1. The user defined structures so dominate the program that, in its unconverted form, it is difficult to recognize it as FORTRAN at all.
2. The original source is free of GO TO statements, thus re-enforcing Dijkstra on the subject of that statement.[11]

REFERENCES

1. Bohm, C. and G. Jacopini, "Flow Diagrams, Turing Machines, and Languages with only Two Formation Rules," *Communications of the ACM*, Volume 9, No. 5, May 1966.

2. Dijkstra, E. W., "Notes on Structured Programming," in Dahl, Dijkstra, and Hoare, *Structured Programming*, Academic Press, 1972.
3. Naur, P., "Programming Languages, Natural Languages, and Mathematics," *Communications of the ACM*, Volume 18, No. 12, December 1975.
4. Ralston, A., Private Communication with this author, June 16, 1975.
5. X3.9.1966 *American Standard FORTRAN*, American Standards Association, Washington, 1966.
6. X3.10.1966 *American Standard Basic FORTRAN*, American Standards Association, Washington, 1966.
7. Wirth, N., "On the Design of Programming Languages," *Proceedings of the IFIP Congress 1974*, North Holland, 1974.
8. Johe, J. M., "Comments on the Topic 'Programming, and Its Implications on Programming Languages," *ACM '75 Proceedings of the Annual Conference.*
9. Archibald, J. A., Jr. and M. Katzper, "On the Preparation of Computer Science Professionals in Academic Institutions," *AFIPS Conference Proceedings*, Volume 43, 1974.
10. Dijkstra, E. W., "Go To Statement Considered Harmful," *Communications of the ACM*, Volume 11, No. 3, March 1968.
11. Ledgard, H. F. and M. Marcotty, "A Genealogy of Control Structures," *Communications of the ACM*, Volume 18, No. 11, November, 1975.
12. Fodor, J. A., T. G. Bever and M. F. Garrett, *The Psychology of Language*, McGraw-Hill Book Co., 1974.

GPMX—A portable general purpose macro processor adapted for preprocessing FORTRAN

by ROBERT C. GAMMILL
The Rand Corporation
Santa Monica, California

ABSTRACT

GPMX is an extension of GPM, a simple, elegant yet powerful language independent macro processor described by Strachey.[1] Unextended, GPM is not suited for preprocessing languages which use column position and end of record to delimit statements. Examples are FORTRAN and many assembly languages. Many programmers are constrained to work in such limited languages and GPMX is a simple yet powerful tool for extending and modifying these languages. Others[6] have developed preprocessors dedicated to a particular language. This has advantages for the implementor, but requires the user to learn a different preprocessor for each language he uses. GPMX is designed to work on any language so that the (non-trivial) effort of learning to use it need not be repeated later. Extensions in GPMX include macro control over: files, record input and output, spacing, conditional macro processing and compilation, access to input and output buffers, and dynamic changing of the macro flag characters. Most of the extensions are accomplished simply by putting the control information on the macro stack where the processor has access to it (ala von Neumann). GPMX has been implemented in ANS FORTRAN for portability. Several applications are shown, including GO-TO free control structures for FORTRAN. Source is available.

GOALS AND PHILOSOPHY

An important goal of this work was the development of a flexible software tool. To be called a tool (using a restrictive definition) software must be suitable for a variety of tasks, on the scale of a single human being (the user) and not dependent upon any unusual aspect of the environment (i.e., adaptable). To make an analogy, a Boeing 747 would not qualify as a tool, (under this definition) for although it is suitable to some tasks, it is not at a human scale (e.g., in terms of maintenance or costs) nor is it adaptable to different environments (e.g., small airports, short hauls, low load levels). However, a crescent wrench would qualify as a tool, since it is suitable to many tasks, at a human scale (e.g., purchasable, maintainable and portable) and can be used in almost any environment (e.g., under water). Many language processors do not qualify as tools. They have been designed for a specific task and environment, so when the situation changes they prove to be clumsy or useless. Then, the large size of most language processors makes them difficult to modify or adapt to the new situation. The lack of adaptable tools has made software development needlessly complex.

GPM qualifies as a tool. It is suitable to a number of tasks and on a human scale. The original implementation of GPM in CPL was extremely compact, and was listed in the last three pages of Reference 1. Gries[5] suggests its implementation as a student exercise (page 433). GPMX appears, to the untrained eye, to be unchanged from GPM. This is because the extensions have been carefully designed to minimize the structural change in GPM. Most of the changes have been accomplished by adding a few primitive macros and by putting the processor input-output and control information on the stack as the values of defined macros, available for use and modification. This is a simple application of the von Neumann stored program concept, and it provides the same startling increase of power to GPM that it provided to the early computers. Finally GPMX has been implemented in ANS FORTRAN for portability.

Although GPMX is a tool, we do not want to imply that it is simple for an inexperienced programmer to learn to use all its power. Simple applications of GPMX can be easy to understand, but an application that uses its full power can be extremely difficult to follow. Perhaps it is best to characterize GPMX as a tool for the advanced scientific or systems programmer, who finds it necessary to write programs in limited languages (such as FORTRAN and assembly) because of special properties they possess, yet desires to have a flexible and powerful macro processor for manipulating his source text.

GPMX provides the following capabilities:

(a) Conditional compilation—so alternative versions of a subroutine need not require multiple source files.
(b) Macro processing—allowing insertion of the same declarations in many subroutines or selective use of open or closed code.

(c) Text compacting—allowing reduction of blanks and other unused redundancy.
(d) Syntax extension—such as adding GO-TO free control constructs to FORTRAN.
(e) Language independence—since information about the language processed is carried in macros not in the processor.
(f) Power and flexibility—the elegance and simplicity of GPM has been kept while its considerable power has been increased.

DESCRIPTION OF GPM

In recent years a number of language independent macro processors have been introduced.[2,3,4] GPM[1] is one of these. In this section we will briefly survey the characteristics of GPM. GPM uses marker characters to indicate where a macro call begins "[", ends "]" and the separations between arguments ":". GPM has only six primitive (built-in) macros. An example showing the use of three of these is:

```
[DEC:[BAR:+:[BIN:3]:[BIN:2]]]     (1)
```

GPM has two data types, "binary" integer and character string. BIN converts from character string to integer and DEC does the reverse. BAR allows integer arithmetic (+−*/). Example (1) produces the character string "5" as its result. The most important primitive macro DEF allows definition of new macros.

```
[DEF:+:<[DEC:[BAR:+:[BIN:%1]:>
                  <[BIN:%2]]]>]     (2)
```

A call on the defined name "+" causes evaluation of its body (second argument of DEF).

```
[+:3:2]     (3)
```

Example (3) will also produce "5" as its result. Example (2) introduced string quotes (< and >) and formal parameters (i.e., %1). Evaluation of nested quotes removes the outermost pair. Macro bodies are normally protected from evaluation at definition time by quoting them. Access to the body of a macro without evaluation is provided by VAL.

```
[VAL:+]     (4)
```

Example (4) will return the body of the + macro, the quoted string from example (2). UPDATE stores new values in the bodies of macros.

```
[UPDATE:+:<This is a new body for +.>] (5)
```

The marker character set being used here is different from that used in GPM.[1] GPMX allows these to be changed at will, so their selection is dictated only by esthetics and local constraints.

An important feature of GPM is that macro calls are allowed anywhere. For example, [[A]:[B]] is a call on a macro whose name is the result of [A] and whose argument is the result of [B]. Also, since macro definition is done by a macro call, macro definitions may be created in unusual ways. One use of this in GPM is the temporary macro definition. If a macro definition is created in the argument sequence of a macro call, that definition will exist only so long as the called macro is being evaluated.

```
[A:[DEF:N:[BIN:3]]]     (6)
```

In example (6) the macro N will be defined (with value 3) only so long as macro A is being evaluated. If there was a previous definition for N, it will be temporarily superseded by this one. One of the important uses of temporary definitions is for self-defining macros, where the definition of the macro occurs in its own argument sequence.

```
[DEF:EVAL:<[X:[DEF:X:%1]]>]     (7)
```

Every time the macro EVAL is called, the macro X is temporarily defined and then called. Since a macro body is evaluated when its name is called, this means that the result of a call on EVAL will be the result of evaluation of its first argument. Using EVAL to cause evaluation and string quotes to defer it gives the ability to specify when evaluation is to be carried out. Thus, although Brown[4] characterizes GPM as passing parameters "by value", as shown in (10), it is possible to pass them "by name", as shown in example (9).

```
[DEF:N:3]
[DEF:A:<[UPDATE:N:[+:[VAL:N]:1]]>
            <[EVAL:%1]>]          (8)
[A:<[VAL:N]>]  <BY NAME>          (9)
[A:  [VAL:N]  ]  <BY VALUE>      (10)
```

Another important result from the ability to create temporary definitions, is that a conditional macro can be defined.

```
[DEF:COND:<[%1:[DEF:%1:%4]>
              <[DEF:%2:%3]]>]     (11)
```

The COND macro depends upon the fact that the most recent definition is used if two macros have the same name. COND executes its third argument if its first and second arguments match, and its fourth argument otherwise. Thus, [COND:A:B:C:D] will produce result D, while [COND:A:A:C:D] will product result C. Conditional execution gives access to the recursive calling capability of GPM by allowing termination of recursive loops. The standard example is the recursive computation of the factorial function (fact(i) := if i=0 then 1 else i*fact (i−1)).

```
[DEF:FACT:<[COND:%1:0:1:<[*:>%1<:>>
      <<[FACT:[−:>%1<:1]]]>]>]     (12)
```

A more practical example would show how data tables may be generated using recursion, but such examples become extremely involved.

EXTENSIONS IN GPMX

In order to make GPM useable as a FORTRAN preprocessor, new capabilities have been added. The elegance and simplicity of GPM makes this easy. In adding new capabilities to GPM care was taken to minimize the set of additional primitive macros, as the elegance of the original set is one of GPM's best features.

The most obvious area of need when using GPM as a FORTRAN preprocessor results from the fact that FORTRAN is column and line (record) oriented while GPM deals with a character string. The solution implemented in GPMX is the introduction of eleven new primitive macros which cope with the problems of record input, output and blank fill. Table I gives the macros and the output or side effect yielded by execution of each.

These new primitive macros use a special marker character for bevity and so as not to conflict with user defined macros which have one character names. In other words, [5] will not give the same result as $5, but will seek a user defined macro named 5. An early version of GPMX used [R] for $1 and [W] for $2, but access to these was often destroyed by subsequent user macro definitions. Thus, $0 through $9 are primitive macros whose meanings cannot be superseded. The user of GPMX who does not need line and column control and fears that a $ macro may occur in the input text, can disable this feature by changing the definition of the $ character to an illegal value. The utility of very short macros in simple text substitution is apparent, so any GPMX macro with a single character name (not a digit) and no arguments can be called in this manner. For example, the macro [DEF:S:<$2$6>], which can be called either as [S] or $S, is useful for moving to column 7 of the next line.

Another mechanism which has been provided in GPMX is the line distributor, which controls the distribution of GPMX input records. We are going to want to make occasional use of GPMX macros in large existing FORTRAN programs. However, scanning every character of a large program which contains only sparse occurrences of macros will be very expensive. The line distributor allows specially marked GPMX lines to be inserted in ordinary FORTRAN, so that only the marked lines need be scanned. Column 1 of each line is reserved for this purpose. The character punched in column 1 controls the handling of the line, much the way a C in column 1 of a FORTRAN line specifies it to be a comment. However, in this extension to GPM, the mode of a character may be set to any desired value (from 1 to 6), allowing succeeding lines starting with that character to be treated in a specified manner. Table II gives the mode values and actions. Default mode value of all characters is 2 except for "–" which has an initial mode value of 6. The line distributor mechanism has several uses. One of these is to provide GPMX comment lines (mode 2). Another is conditional compilation. Putting a character in mode 4 will cause all lines with that character in column 1 to be compiled. If that character is set to mode 3 on a subsequent pass, all those lines will become FORTRAN comments. GPMX lines can produce widely varying numbers of output characters, and if blanks are taken as significant, much of the output becomes unwanted blanks. Thus, it was decided in GPMX to ignore unquoted blanks in the input. This is only done on mode 6 lines, which are passed to the GPMX scanner. For those who use GPMX with another language, the column position of the line distributor control character can be changed to any desired value, through use of the UPDATE macro. Also, should the user desire to disable the line distributor mechanism, and operate in GPM mode (scanning every line), that too can be accomplished through macro calls.

Besides the primitive line format control macros, several other primitive macros have been added. These are listed below. Each has been chosen to be of general use. The second through fourth macros are the character string primitives of PL/1.

(a) [SETMODE:[BIN:5]:< C0123456789>] sets mode values of line distributor control characters. In this case the characters normally found in column 1 of a FORTRAN line are being set to mode 5.

(b) [LENGTH:[VAL:S]] returns a value (binary integer) which is the length of its argument. In this case the result will be 4 (see preceding DEF of S).

(c) [SUBSTR:<ABCDEFG>:[BIN:3]:[BIN:2]]

TABLE I—Semantics of Special $ Macros

MACRO	OUTPUT	SIDE EFFECT
$0	none	dumps the run-time stack
$1	none	reads next GPMX input record
$2	none	writes next GPMX output record
$3	3 blanks	none
$4	4 blanks	none
$5	5 blanks	none
$6	6 blanks	none
$7	7 blanks	none
$8	8 blanks	none
$9	9 blanks	none
$$	$	none

TABLE II—Line Distributor Actions Specified by Character Modes

Mode	Action	Explanation
1	Reject	No input or output listing.
2	Reject	Input list but no output listing.
3	Output	Set column 1 to C.
4	Output	Set column 1 to blank.
5	Output	Leave line unchanged.
6	Scan	GPMX scan begins at column 2.

returns a substring of its first argument. The second argument tells where the substring begins and the third tells how long it is. The result is CD.

(d) [INDEX:<ABCDEFG>:<CDE>] returns a binary integer which gives the position where its second argument first occurs as a substring of its first. In this case the result will be a binary 3. If there are no occurrences the result is zero.

(e) [WEF:[BIN:7]] writes an end-of-file on logical unit 7.

(f) [REW:[BIN:8]] rewinds logical unit 8.

Besides the addition of new primitive macros and the line distributor mechanism, a very important change has been made in GPMX. This involved moving most of the important control variables, switches and buffers to the stack, as ordinary GPMX macros whose values may be changed by UPDATE. This is a very important improvement. For example, it allows the macro flag characters to be changed during macro processing, and with no new support mechanism required.

```
 * CHANGE CONTROL CHARACTERS
 – [UPDATE:CONCHR:<@,;?()&>&1
 * SET THEM BACK
 – @UPDATE,CONCHR,([:]%<>$);$1
```

Another important use of this facility is changing logical unit numbers for the various input and output streams by means of UPDATE. Logical unit number zero has been taken to mean that no I/O should occur on the particular stream. For example, the following statements will turn on the output listing and turn off the output, for testing without compilation of the output.

```
– [UPDATE: OUTPUT: [BIN:0]]
– [UPDATE: OUTLST: [BIN:6]]
```

The ability to change the logical unit numbers for the input and output, in conjunction with the primitive macros WEF and REW, allows the GPMX programmer complete control over input and output files. Partitioning of output into two files, one for declarations and another for executable code (a frequently needed capability in a preprocessor) is easy to program at the macro level. Also, multiple passes through the input text can be carried out using these facilities.

Other important GPMX control elements can also be changed using UPDATE or accessed using VAL. One particularly important macro, COLUMN, contains the integer which tells in what column the next output character will go. This is especially useful when writing macros to format the output correctly (e.g., tabbing). The contents of the input buffer, INBUF, can be manipulated as a character string (using INDEX, SUBSTR, VAL and UPDATE) before any macro scanning takes place, so GPMX can be used as a general text processor with controlled input. This is a capability that Brown[7, page 51] specifically notes as missing from GPM, when comparing it with the TRAC language.

GPMX APPLICATIONS

In this section we will examine some GPMX applications. In all cases the examples deal with FORTRAN. This was done because FORTRAN is widely known and because it involves most of the unpleasant problems that will be encountered by a language independent preprocessor. The first example shows the generation of local variables and statement numbers as they can be applied in a DO macro. This example uses the fact that a macro definition (DEF) which is created as part of the argument sequence of another macro will exist only as long as that macro is being evaluated. This allows temporary variables to be created. Here K is used to store the local variable name and M is used to store the local statement number. (Note: %%1 means the first argument of the caller's caller).

```
 * GENERATE STATEMENT NUMBER
 * OR INTEGER VARIABLE
 – [DEF:GENSN:<[DEF:%1:<77>>
                <[DEC:[GEN]]]>]
 – [DEF:GENIV:<[DEF:%1:<IV>>
                <[DEC:[GEN]]]>]
 – [DEF:B:[BIN:1]]  [DEF:N:[BIN:10]]
 * GEN RETURNS THE VALUE
 * OF N AND INCREMENTS N.
 – [DEF:GEN:<[VAL:N][UPDATE:N:>
              <[BAR:+:$N:$B]]>]
 * EVAL MACRO EVALUATES ITS
 * ARGUMENT.
 – [DEF:EVAL:<[8:[DEF:8:%1]]>]
 * CONT MACRO PRODUCES A CONTINUE
 * CARD
 – [DEF:CONT:<$2[%1]< CONTINUE>>]
 * DO MACRO
 – [DEF:DO:<[DOBODY:[GENIV:K]>
 –             <[GENSN:M]]>]
 – [DEF:DOBODY:<DO $M $K=1,%%1>
 –                 <[EVAL:%%2][CONT:M]>]
 * CALL THE DO MACRO
 – $S [DO:<32>:<$SB($K)=A($K)>]
```

The preceding code yields:

```
  DO 7711 IV10=1,32
  B(IV10)=A(IV10)
7711 CONTINUE
```

In this example the body of the DO is not evaluated until it is passed into the interior of the DO macro. This is an example of a useful application of passing arguments "by name." Here it allows the temporary macro K to carry the local name for the integer DO variable. A disadvantage of this example is that the complete body of the DO must be passed as the second argument of the DO macro, enclosed in string quotes. This would be extremely awkward for long DO loops.

One of the goals set for GPMX was syntax extension of FORTRAN, to allow GO-TO-free constructs such as if-then-else and while-do to be introduced into the language. Initially we tried to do this in a manner similar to that shown in the preceding example for DO loops. This proved awkward, since the body of code in each part of an if-then-else had to be passed as an argument to a macro, tending to overload the stack. Furthermore, it is very desirable to be able to insert these control statements within ordinary FORTRAN code in the following manner:

```
– [IF] I. NE. 3 .AND. F(I,J).GT. 3. 4 [THEN]
        a sequence of FORTRAN statements
–[ELSE]
        another sequence of FORTRAN statements
–[FI]
```

To implement the preceding idea, we create a special macro whose body will serve as a short push-down stack for statement labels. Then we define macros to allow us to PUSH and POP this special stack, as well as one to allow us to pick up a copy of the TOP element (without changing the stack).

```
– [DEF:PUSH:<[UPDATE:%2:%1[SUBSTR:>
–                   <[VAL:%2]:[BIN:1]:>
–              <[BAR:–:[LENGTH:[VAL:%2]]:>
–                   <[BIN:1]]]]>]

–  [DEF:POP:   <[UPDATE:%1:[SUBSTR:>
–                   <[VAL:%1]:[BIN:2]:>
–              <[BAR:–:[LENGTH:[VAL:%1]]:>
–                   <[BIN:1]]]< >]>]

–  [DEF:TOP:   <[SUBSTR:[VAL:%1]:>
–                   <[BIN:1]:[BIN:1]]>]
```

Using the preceding macros, we implement the if-then-else-fi construct as four separate macros, in the following manner. The definition for GEN was given earlier in this section.

```
–[DEF:IF:<$S<IF(.NOT.(>>]
– [DEF:G:<<GO TO >>]

–[DEF:THEN:<<)) >$G[PUSH:[GEN]:>
–          <STACK]<77>[DEC:[TOP:STACK]]>]

–[DEF:ELSE:<$S$G<77>[DEC:[VAL:N]]>
–                <$2<77>[DEC:[TOP:STACK]] >
–          <[UPDATE:STACK:[GEN]:[BIN:1]]>]
–[DEF:FI:<$B<77>[DEC:>
–            <[TOP:STACK]] [POP:STACK]>]
–[DEF:STACK:$9]
–[DEF:S:<[COND:[BIN:7]:>
–          <[VAL:COLUMN]::<$2$6>]>]
– [DEF:B:<[COND:[BIN:7]:>[VAL:COLUMN]
–            <<CONTINUE>$2>:<$2>]>]
```

In this example we have introduced new macros S and B for controlling our position in the output card image. $S moves us to column 7 of the next line, if we are not already there. $B moves us to column 1 of the next line, adding a CONTINUE statement if we are presently in column 7 of a line. These two macros demonstrate how we can use COLUMN (the position in OUTBUF where the next output character goes) to control positioning of output information.

As we used GPMX on actual programs, written in structured FORTRAN, it became clear that our macro definitions for GO-TO-free FORTRAN had still not adequately solved all the problems. As a result, we changed from a single stack to double stacks, one containing the label desired for exit from a control structure and the other containing the label desired when the next cycle (pass) through a control structure is to be started. The EXIT and NEXT macros, which generate FORTRAN GO-TO's, require a first argument which specifies how far out in a nested control structure the jump is to go. We also implemented automatic indentation of the output text (as dictated by the nested control structure) and added most of the commonly mentioned control structures (e.g. CASE, REPEAT, WHILE, FOR). All of this was carried out completely at the macro level. Figure 1 shows the use of some of these capabilities in a simple program. The library of macros is given in the Appendix.

We now feel that we have achieved a useable set of macros for GO-TO-free FORTRAN. However, many further improvements can still be introduced. We note with satisfaction, however, that although we write and rewrite macros quite often, changes in the processor have become an infrequent event. An example of such an event occurred when moving GPMX to PDP-11 UNIX. It became desirable to allow the GPMX programmer to associate a filename with a logical unit number, providing complete macro time control over input and output files. This required the addition of a new primitive macro which is system dependent (not definable inside FORTRAN). The macro is [SETFILE:[BIN:<digit>]:<filename>]. Code written using this macro acts much like a job control language. It seems likely that with the addition of yet another system dependent primitive macro (EXEC perhaps) allowing execution of assemblers, compilers, loaders and other system utilities, GPMX could be used as an extensible job control monitor for minicomputers.

Recalling our earlier description of a tool, we feel

```
*        DEFINITIONS OF COMMON BLOCKS
- [DEF:TTTINF:<$M/%0/MOVE(64),HOLD(64),IMOVE,MVSEQ(64),SUM(76)>
-             <$&,NPLANE(18)$IHOLD,SUM>]
- [DEF:CTRL  :<$M/%0/ITURN,PLAYER(10,3),IVAL(2),MVCHR(3),>
-             <MV4,MV10$&,NNODES,MODE,BLANK$IPLAYER,BLANK>]
*        MAIN PROGRAM
- [TTTINF] [CTRL] $L DONE,CMPUTR $S <CALL INPUT(1)>
- [REPEAT] $S <CALL INITAL>
-    [REPEAT] $S <CALL INPUT(2)>
-       [REPEAT] $S ITURN=MOD(IMOVE,2)+1
-          [IF] CMPUTR(ITURN) [THEN] $S <CALL STRAT>
-             [ELSE] $S <CALL INPUT(3)>
-                    $S <IF(MV10.LE.0) >[NEXT:3]
-          [FI] $S <CALL ENTER(MV4,MVCHR(ITURN),IVAL(ITURN))>
-          $S <IF(CMPUTR(ITURN)) CALL OUTPUT(5)>
-       [UNTIL] DONE(X) [ENDREP] $S MV4=1
-    [TILNOT] MV4.EQ.2 [ENDREP]
- [TILNOT] MV4.EQ.1 [ENDREP] $S STOP $E

* OUTPUT FROM THE MACRO PROCESSOR.
      COMMON/TTTINF/MOVE(64),HOLD(64),IMOVE,MVSEQ(64),SUM(76)
     *,NPLANE(18)
      INTEGER HOLD,SUM
      COMMON/CTRL/ITURN,PLAYER(10,3),IVAL(2),MVCHR(3),MV4,MV10
     *,NNODES,MODE,BLANK
      INTEGER PLAYER,BLANK
      LOGICAL DONE,CMPUTR
      CALL INPUT(1)
7710     CALL INITAL
7712        CALL INPUT(2)
7714           ITURN=MOD(IMOVE,2)+1
               IF(.NOT.(CMPUTR(ITURN))) GO TO 7716
                  CALL STRAT
                  GO TO 7717
7716              CALL INPUT(3)
                  IF(MV10.LE.0) GO TO 7713
7717           CALL ENTER(MV4,MVCHR(ITURN),IVAL(ITURN))
               IF(CMPUTR(ITURN)) CALL OUTPUT(5)
7715        IF(.NOT.(DONE(X))) GO TO 7714
            MV4=1
7713     IF((MV4.EQ.2)) GO TO 7712
7711  IF((MV4.EQ.1)) GO TO 7710
      STOP
      END
```

Figure 1—Use of GO-TO free control structures with FORTRAN

that GPM and its offspring GPMX are definitely tools. As we have indicated, with minor modifications these macro processors seem capable of use as general text processors, as language translators or as job control monitors. Of course, the limitations must be recognized (as when racing bicycles against cars) but the flexibility and simplicity seems astonishing.

ASSESSMENT

GPMX is an adaptable tool, easily modified or controlled by a sophisticated user. It has been moved to a wide variety of computers (including minis) in a few man-hours. A disadvantage of implementation in ANS FORTRAN is that handling card images via formatted I/O is inefficient, although essential for portability. However, the I/O in GPMX is localized in a few routines, so substitution of non-portable but efficient I/O is easy.

GPMX is dramatically different from other FORTRAN preprocessors, since it is a general purpose macro processor extended for that use. GPM was not designed for ease of use by unsophisticated FORTRAN applications programmers. Thus, preprocessors (such as MORTRAN2[6]) which were designed for that goal, tend to be more suitable for use by FORTRAN programmers unfamiliar with general purpose macro processors. However, GPMX has substantially more computational power than most other FORTRAN preprocessors, and it is language independent besides. Although MORTRAN2 and some other FORTRAN preprocessors also have macro capabilities, in most cases this is of the pattern match and simple text substitution variety. None of these have sufficient computational power to implement stacks or source symbol generation at the macro level, as demonstrated here. These advantages make GPMX especially suitable for use by advanced applications and systems programmers, for whom FORTRAN preprocessing is only one of the desired applications. A special advantage enjoyed by GPMX is that most extensions can be produced by defining new macros, while extensions in other preprocessors often involve rewriting the processor. The most obvious disadvantage of GPMX (like GPM) is its ugly syntax. The many good features should overcome this irritation in the appropriate applications.

REFERENCES

1. Strachey, C., "A General Purpose Macrogenerator," *Computer Journal* 8,3, October 1965, pp. 225-241.
2. Waite, W. M., "A Language-Independent Macroprocessor," *CACM 10*, 7, July 1967, pp. 433-440.
3. Halpern, M. I., "XPOP: A Meta-Language Without Metaphysics," *Proc. AFIPS*, 1964 FJCC, Vol. 26, pp. 57-68.
4. Brown, P. J., "A Survey of Macro Processors," *Annual Review in Automatic Programming*, Vol. 6, Part 2, 1969, pp. 37-88.
5. Gries, D., *Compiler Construction for Digital Computers*, John Wiley and Sons, New York, 1971.
6. Cook, A. James, *MORTRAN2 Reference Manual*, Computation Research Group, Stanford Linear Accelerator Center.
7. Brown, P. J., *Macro Processors and Techniques for Portable Software*, John Wiley and Sons, New York, 1974.

APPENDIX

```
*      GENERATE A LOCAL STATEMENT NUMBER.
- [DEF:NXTNAM:[BIN:10]]$1 INITIALIZE INTEGER FOR LOCAL NAMES
- [DEF:INCR:<[UPDATE:%1:[BAR:+:[VAL:%1]:>[BIN:1]<]]>]$1
- [DEF:GEN:<[VAL:NXTNAM][INCR:NXTNAM]>]$1  GENERATE A NUMBER
*       DEFINE LINE AND SPACING CONTROL MACROS S, B AND Z.
- [DEF:S:<[COND:>[BIN:7]<:[VAL:COLUMN]:<$Z>:<$2$6$Z>]>]$1
- [DEF:B:<[COND:>[BIN:7]<:[VAL:COLUMN]:<$Z$C$2>:<$2>]>]$1
- [DEF:Z:$9$9$9] [UPDATE:Z:] $1 Z INITIALLY EMPTY STRING
- [DEF:DNZ:<[UPDATE:Z:[SUBSTR:$Z:>$1
-           [BIN:1]<:[BAR:-:[LENGTH:$Z]:>[BIN:3]<]]]>]$1
- [DEF:UPZ:<[UPDATE:Z:$Z$3]>] $1
*        DEFINE MACROS FOR FORTRAN TO HANDLE KEY-WORDS.
- [DEF:G:<<GO TO >>][DEF:C:<CONTINUE>][DEF:M:<$S<COMMON>>]$1
- [DEF:Q:<$S<EQUIVALENCE >>][DEF:I:<$S<INTEGER >>]$1
- [DEF:U:<$S<SUBROUTINE >>][DEF:R:<<RETURN>>][DEF:F:<<FORMAT>>]
- [DEF:D:<$S<DATA >>][DEF:E:<$S<END>$2>][DEF:&:<$2$5*$Z>]$1
- [DEF:#:<<IF(.NOT.(>>] [DEF:@:<<IF((>>][DEF:L:<$S<LOGICAL >>]
*           DEFINE STACK HANDLING MACROS.
- [DEF:PUSH:<[UPDATE:%2:%1[SUBSTR:[VAL:%2]:>[BIN:1]<:[BAR:-:>
-          <[LENGTH:[VAL:%2]]:>[BIN:1]<]]]>]$1 PUT ITEM ON TOP
- [DEF:POP:<[UPDATE:%1:[SUBSTR:[VAL:%1]:>[BIN:2]<:[BAR:-:>
-          <[LENGTH:[VAL:%1]]:>[BIN:1]<]]< >]>]$1 DISCARD TOP
- [DEF:GET:<[SUBSTR:[VAL:%1]:[BIN:%2]:>[BIN:1]<]>] $1 GET ITEM
- [DEF:PUT:<[UPDATE:%2:%1:[BIN:%3]]>] [DEF:TOP:<[GET:%1:1]>] $1
- [DEF:STACK:$9] [DEF:STCKX:$9] $1 MAKE STATEMENT NUMBER STACKS.
*        DEFINE CONTROL STRUCTURE MACROS FOR GO-TO FREE FORTRAN.
- [DEF:IF    :<$S$#[PUSH:0:STCKX][PUSH:I:STACK][UPZ]>] $1
- [DEF:THEN  :<[PUT:[GEN]:STACK:1]<)) >[NEXT:1]>] $1
- [DEF:ELSE  :<$S[EXIT:1][NLBL][PUT:[TOP:STCKX]:STACK:1]>] $1
- [DEF:FI    :<[COND:[TOP:STACK]:[TOP:STCKX]::<[NLBL]>]> $1
-             <[XLBL][POP:STACK][DNZ]>] $1 END OF IF
- [DEF:REPEAT:<$B<77>[DEC:[GEN]]  [UPZ][PUSH:[GEN]:STACK]> $1
-             <[PUSH:0:STCKX]>] $1
- [DEF:UNTIL :<[DNZ][NLBL]$Z$#>] $1
- [DEF:TILNOT:<[DNZ][NLBL]$Z$@>] $1
- [DEF:ENDREP:<<)) >[LPBACK]>] $1
- [DEF:LPBACK:<$G<77>[DEC:[BAR:-:[TOP:STACK]:>[BIN:1]<]]> $1
-             <[POP:STACK][XLBL]>] $1
*        DEFINE THE EXIT AND NEXT FUNCTIONS AND LABEL MACROS.
- [DEF:EXIT:<[COND:[GET:STCKX:%1]:0:<[PUT:[GEN]:STCKX:>%1<]>:]>
-           <$G<77>[DEC:[GET:STCKX:%1]]>] $1 EXIT THIS STRUCTURE
- [DEF:XLBL:<[COND:[TOP:STCKX]:0::<$B<77>[DEC:[TOP:STCKX]]  >]>
-           <[POP:STCKX]>] $1 GENERATE EXIT LABEL IF NOT 0.
- [DEF:NEXT:<$G<77>[DEC:[GET:STACK:%1]]>] $1 GO TO NEXT CASE
- [DEF:NLBL:<$B<77>[DEC:[TOP:STACK]]  >] $1 LABEL FOR NEXT CASE
```

An experiment comparing Fortran programming times with the software physics hypothesis

by R. D. GORDON and M. H. HALSTEAD
Purdue University
Lafayette, Indiana

ABSTRACT

Recent discoveries in the area of Algorithm Structure or Software Physics[1-25] have produced a number of hypotheses. One of these relates the number of elementary mental discriminations required to implement an algorithm to measurable properties of that algorithm, and the results of one set of experiments confirming this relationship have been published.[16] That publication, while significant, made no claim to finality, suggesting instead that further experiments were warranted. This paper will present the results of a second set of experiments, having the advantages of being conducted in a single implementation language, Fortran, from problem specifications readily available in computer textbooks.

The first section of this paper presents the timing hypothesis, and the elementary equations upon which it rests. The second section presents the details of the experiment and the results which were obtained, and the third section contains an analysis of the data.

TIMING HYPOTHESIS

Measurable properties of any implementation of any algorithm include:

η_1 = The count of distinct operators
η_2 = The count of distinct operands (variables or constants)
N_1 = Total uses of operators
N_2 = Total uses of operands

The vocabulary, η, is given by:

$$\eta = \eta_1 + \eta_2 \tag{1}$$

and the length, N, is:

$$N = N_1 + N_2 \tag{2}$$

From these properties, it is possible to obtain the volume, V, in bits, as:

$$V = N \log_2 \eta \tag{3}$$

and the implementation level, L, where $L \leq 1$, as:

$$L = \frac{\eta_1^*}{\eta_1} \frac{\eta_2}{N_2} \tag{4}$$

where η_1^*, the minimum possible number of operators, will equal 2 for most algorithms. (One for the name of a function, plus one for a grouping symbol operator). It has been shown[4] that the product $L \times V$ is invariant under translation from one language to another, and that for programs without impurities:[3,6,8]

$$N = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 \tag{5}$$

From this point, the following nine steps yield the timing equation:

1. A program consists of N selections from η elements.
2. A binary search of η elements requires $\log_2 \eta$ comparisons.
3. A program is generated by making $N \log_2 \eta$ comparisons.
4. Therefore, the volume, V, is a count of the number of comparisons required.
5. The number of elementary mental discriminations required to complete one comparison measures the difficulty of the task.
6. The level, L, is the reciprocal of the difficulty.
7. Therefore, E, the count of elementary mental discriminations required to generate a program, is given by:

$$E = \frac{V}{L} \tag{6}$$

8. S, the speed with which the brain makes elementary mental discriminations can be obtained from psychology[26] as:

$5 \leq S \leq 20$ discriminations per second.

9. Therefore, the time to generate a *preconceived* program, by a *concentrating* programmer, *fluent* in a language, is:

$$\hat{T} = \frac{V}{SL} \tag{7}$$

Equation 7 may be expressed in more basic terms by substituting for V from equation 3, and for L from equation 4, with $\eta_1^* = 2$, giving:

$$\hat{T} = \frac{\eta_1 N_2 N \log_2 \eta}{2\,S\,\eta_2} \tag{8}$$

The effect of possible impurities[5] may be eliminated from equation 8 by substituting for N from equation 5. Letting $S=60\times 18=1080$ will then give, for time in minutes:

$$\hat{T}=\frac{\eta_1 N_2(\eta_1 \log_2\eta_1+\eta_2 \log_2\eta_2)\log_2\eta}{2160\,\eta_2} \tag{9}$$

Each of the variables on the right hand side of equation 9 can be readily measured (or counted) in any computer program, and the experiment described in the next section was designed to compare results from that equation with observed programming times.

EXPERIMENTAL PROCEDURE

Eleven problems were arbitrarily selected from two published sources. In selecting candidates for the experiment, problems were sought which were stated in a non-procedural form. Further, the problem statement had to be complete. That is, in the course of solving a particular problem, specific laws of physics, mathematics, etc. would not have to be derived. The problems finally selected were taken from Knuth,[27] and from Maurer and Williams,[28] and cover a wide range of topics including character manipulation, list processing, simulation experiments and mathematical analysis. The source of each problem statement is cited in Table I.

On each of eleven days, one of these problems was implemented by the senior author. In order to maintain a consistent level of performance all work was conducted in a quiet room, free from distractions, during the same period of the day. The time required to fully implement the problem was obtained. This total time included the number of minutes spent reading the statement of the problem, preparing flowcharts and writing preliminary versions of the code, writing the final version of the code, desk checking, and the time spent working to correct errors in the program. Time to keypunch was not included.

For a number of reasons, including availability and fluency, all of the algorithms were implemented in Fortran. In the course of solving a problem the correctness of the implementation was checked by executing a sufficiently complex test case for which a correct answer was known. In some cases the solution to a problem was written as a subroutine and testing required that a main routine be written. In such a case only the preparation of the subroutine was considered for the experiment. In addition, several implementations made use of subroutines previously written. Such routines were also not included.*

After each program was completed, a careful count was made to determine values of η_1, η_2, N_1 and N_2. In obtaining these values all read, write, declarative statements and comments were ignored. The results are shown in Table I.

TABLE I—Experimental Data

	Program Specifications			Software Parameters				Implementation
No.	Ref.*	Page	Problem	η_1	η_2	N_1	N_2	Time-Minutes
G1	K	158	21	15	11	59	51	19
G2	K	159	23	20	24	231	197	92
G3	K	196	7	16	12	64	49	16
G4	K	377	17	19	21	131	113	39
G5	K	158	22	7	10	38	35	21
G6	K	154	10	9	14	69	62	30
G7	M	32	3.2.21	12	8	30	23	5
G8	M	32	3.2.23	19	15	73	55	24
G9	M	88	8.3.2	22	32	124	104	43
G10	M	89	8.3.4	25	34	261	222	91
G11	M	27	3.2.4	14	10	29	21	5

* K=Knuth [27], M=Maurer and Williams.[28]

ANALYSIS OF THE DATA

The programming time predicted by theory was obtained for each program by applying equation 9 to the data in Table I. This result, T, can be compared with the observed value, T, in Table II. In addition, a count of the number of statements in each program was obtained, and the programs were ordered according to these values.

The average of the calculated values, 34 minutes, is fortuitously close to the observed value, 35 minutes. The coefficient of correlation is 0.934, only slightly smaller than the value of 0.952 reported in an earlier experiment.[16] In further agreement with that experiment, the correlation between length and observed times, 0.887, is lower than between observed and calculated times.

In conclusion, it may again be observed that one more set of experimental data does not contradict the simple hypothesis. As a result, further carefully controlled experiments by others would appear to be warranted.

* Additional details available from the author.

TABLE II—Experimental Results

Program Number	Statement Count	Programming Time-Minutes	
		T observed	T Equ. 9
G7	7	5	4.6
G11	8	5	5.4
G5	11	21	2.5
G6	15	30	6.8
G3	18	16	15.6
G1	18	19	14.6
G8	18	24	22.9
G4	32	39	43.6
G2	36	92	81.5
G9	38	43	49.2
G10	59	91	128.5
Means		35.0	34.1

REFERENCES

1. Bayer, Rudolf, *A Theoretical Study of Halstead's Software Phenomenon,* CSD Tech. Rept. No. 69, Purdue, May 1972.
2. Bayer, Rudolf, *On Program Volume and Program Modularization,* CSD Tech. Rept. No. 105, Purdue, September 1973.
3. Bohrer, Robert, "Halstead's Criteria and Statistical Algorithms," *Proc. 8th Computer Science/ Statistics Interface Symposium,* Los Angeles, February 1975.
4. Bulut, Necdet, *Invariant Properties of Algorithms,* Ph.D. Thesis, Purdue, August 1973.
5. Bulut, Necdet and M. H. Halstead, "Impurities Found in Algorithm Implementations," *ACM SIGPLAN Notices,* 9, 3, March 1974.
6. Bulut, Necdet, M. H. Halstead and Rudolf Bayer, "The Experimental Verification of a Structural Property of Fortran Programs," *Proc. ACM Annual Conference,* San Diego, 1974.
7. Funami, Yasao and M. H. Halstead, *Software Physics Analysis of Akayama's Debug Data,* CSD Tech. Rept. No. 144, Purdue, May 1975.
8. Halstead, M. H., "Natural Laws Controlling Algorithmic Structure," *ACM SIGPLAN Notices,* 7, 2, February 1972.
9. Halstead, M. H., *A Theoretical Relationship Between Mental Work and Machine Language Programming,* CSD Tech. Rept. No. 67, Purdue, May 1972.
10. Halstead, M. H. and Rudolf Bayer, "Algorithm Dynamics," *Proc. ACM Annual Conference,* Atlanta, 1973.
11. Halstead, M. H., "An Experimental Determination of the 'Purity' of a Trivial Algorithm," *ACM SIGME Performance Evaluation Review,* 2, 1, March 1973.
12. Halstead, M. H., "Language Level, A Missing Concept in Information Theory," *ACM SIGME Performance Evaluation Review,* 2, 1, March 1973.
13. Halstead, M. H. and P. M. Zislis, *Experimental Verification of Two Theorems of Software Physics,* CSD Tech. Rept. No. 97, Purdue, June 1973.
14. Halstead, M. H., *Software Physics Comparison of a Sample Program in DSL ALPHA and COBOL,* IBM Research Report No. RJ 1460, October 1974.
15. Halstead, M. H., *Software Physics: Basic Principles,* IBM Research Report No. RJ 1582, May 1975.
16. Halstead, M. H., "Toward a Theoretical Basis for Estimating Programming Efforts," *Proc. ACM Annual Conference,* Minneapolis, 1975.
17. Kennedy, Dale and Roger Bruning, *Childrens Descriptions of Complex Objects,* Report 505-68-6939, Univ. of Nebraska, Lincoln, October 1974.
18. Kulm, Gerald, "Information Content—An Alternative Measure of Reading Complexity," *American Psychological Association Annual Meeting,* New Orleans, August 1974.
19. Ostapko, D. L., "On Deriving a Relation Between Circuits and Input/Output by Analyzing an Equivalent Program," *ACM SIGPLAN Notices,* 8, 6, June 1974.
20. Ostapko, D. L., "Analysis of Algorithms Implemented in Software and Hardware," *Proc. ACM Annual Conference,* San Diego, 1974.
21. Zislis, Paul, *An Experiment in Algorithm Implementation,* CSD Tech. Rept. No. 96, Purdue, June 1973.
22. Zislis, Paul M., "Semantic Decomposition of Computer Programs: An Aid to Program Testing," *ACTA Informatica 4,* pp. 245-269, 1975.
23. Zweben, S. H., *Software Physics: Resolution of an Ambiguity in the Counting Procedure,* CSD Tech. Rept. No. 93, Purdue, April 1973.
24. Zweben, S. H., *The Internal Structure of Algorithms,* Ph.D. Thesis, Purdue, May 1974.
25. Zweben, S. H., "A Recent Approach to the Study of Algorithms," *Proc. ACM Annual Conference,* San Diego, 1974.

Additional References:

26. Stroud, John M., "The Fine Structure of Psychological Time," *Annals of N.Y. Academy of Sciences,* 1966, pp. 623-631.
27. Knuth, Donald, *The Art of Computer Programming,* Addison Wesley Publishing Co., Massachusetts, 1969, Vol. 1.
28. Maurer, H. A. and M. R. Williams, *A Collection of Programming Problems and Techniques,* Prentice-Hall, Inc., New Jersey, 1972.

Representations of networks

by HARVEY J. GREENBERG and JAMES E. KALAN
Virginia Polytechnic Institute and State University
Blacksburg, Virginia

ABSTRACT

Bit map and list structures are analyzed for representation of a network, using its adjacency matrix. Storage analysis, with reduction for m-partite networks, reveals a fundamental function of problem dimensions, called the "index threshold." Several examples, from industry and government, are cited to illustrate the analysis. Relative processing merits are studied, and an equation is derived to relate "processing ratio" to the ratio of the index threshold to the index size. Finally, conversion algorithms are presented, designed to minimize extra work space.

PRELIMINARIES

Let V denote a (finite) set of *vertices,* whose elements are consecutive positive integers. (If names are used, a separate internal structure is employed to identify a vertex by its index number.) Let A denote a set of *arcs,* whose elements are ordered pairs of vertices. If (i,j) is an arc in A, then vertex j is a *successor* of vertex i, and i is a *predecessor* of j.

For any finite set, S, let '/S/' denote its cardinality. For example, /V/ is the number of vertices, and /A/ is the number of arcs. The *adjacency matrix* of a network is a /V/ by /V/, 0-1 matrix with element (i,j) equal to 1 if, and only if, vertex j is a successor of vertex i. Since the adjacency matrix has /A/ nonzero entries, all of which are 1, Kalan's[11] supersparse representation yields a familiar list structure to represent a network.[2] Specifically, define a list (or file) with one record per vertex. The data items in a record are the successors (if stored by rows) or predecessors (if stored by columns). The record size is the (positive) *degree* of the associated vertex (i.e., its number of successors or predecessors, respectively). Note that while the adjacency matrix, per se, cannot represent parallel arcs, its list structure can, if this should be desired.

Another representation of the adjacency matrix is the *bit map,* using one bit per element. For every vertex, its successor list is identified by a contiguous bit string, having fixed length, /V/. Bit j is on if vertex j is a successor of the associated vertex; otherwise, it is off. Garfinkel and Nemhauser[7, p. 304] alluded to the use of bit maps to represent an adjacency matrix, although they did not consider its effect on processing.

Lefkovitz[13] identified the trade-off between bit map and list structures for set representation as ". . . the dilemma for large-scale data bases, . . ." He concluded that bit maps take more space when the lists are short, and less space when they are long.

Orchard-Hays[16, p. 81 and 212] considered, briefly, the use of a bit map in a network-related context, and the next section of this study may be considered a generalization of his threshold analysis.

Pooch and Nieder[17] provided a general survey of these two methods. They also observed the natural correspondence between set operations (e.g., UNION, INTERSECTION) and logical operations (e.g., OR, AND) which render the bit map structure attractive for such processing. However, we shall show how the apparent superiority of bit maps, in the cases cited here, really depends upon the density.

A comprehensive study of the interface between data structures and algorithms was accomplished by MacVeigh.[14] He considered fundamental numerical algorithms with matrices. We have a different, but analogous, situation in dealing with operations on the topology of a network.

It should be noted that analysis of mixed forms of bit map and list structures follows from this study of the two pure forms.

STORAGE ANALYSIS

The bit map structure uses $/V/^2$ bits, with one bit map of length /V/ for each vertex. The list structure uses h/A/ bits, where the *index size,* h, is the number of bits to store an index.

Define the *index threshold* as:

$$H = /V/^2 \div /A/.$$

(This is the reciprocal of the density of the adjacency matrix.) Then, the list structure uses more (less) space if h is greater (less) than H.

If we let h be data-driven, then we may use $h=\lfloor \log_2/V/\rfloor$. Alternatively, we may fix h to represent any network up to 2^h-1 vertices. (The latter choice is akin to current design of LP software in representing row indices.) For fixed h, we consider the range, $15\leqq h\leqq 18$. This permits representation of networks having more than 32,000 vertices, and the values in this range are convenient for most computers (i.e., h=15 for CDC 6,000's, h=16 for IBM 370's and h=18 for UNIVAC 1100's).

Before we examine the relative storage requirements for a variety of applications, note that the index threshold may be expressed in terms of the *average degree*:

$$D=/A/\div/V/.$$

That is, D is the average number of successors per vertex. Then, we note:

$$H=/V/\div D.$$

(This expression shall be useful when we analyze relative processing efficiency.)

Table I below lists problem dimensions reported by Glover and Klingman,[8] together with their index thresholds and their average degrees. The Flight Training problem is sufficiently dense that the bit map structure requires less space, even when $h=\lfloor \log_2/V/\rfloor=10$. The same is true for the first form of the Cotton Gin problem. However, this is reversed for the reduced and compacted forms; like the Treasury and Shipyard problems, the list structure requires less space, even when the index size is fixed at 18.

Actually, the figures in Table I may be misleading because, for most applications, the storage requirements of both structures may be reduced by letting semantics describe part of the topology. Before we explain further, let us consider the transportation problem[5] with the present method.

Suppose we have n suppliers and m demand points, so /V/=n+m. Potentially, there can be nm arcs, one from each supplier to every demand point. However, in practice only a fraction, say f, of these arcs are actually present. This is because a supplier is linked to this fraction of demand points, on the average (or equivalently, the average number of suppliers for a demand point is fn). Then, /A/=fnm, so the index threshold is:

$$H=(n+m)^2/fnm.$$

TABLE I—Sample Problems[8]

Problem	Number of Vertices (/V/)	Number of Arcs (/A/)	Index Threshold (H)	Average Degree (D)
Treasury	5,000	625,000	40.0	125.0
Cotton Gin				
first form	4,200	2,460,000	7.2	585.7
reduced	5,141	95,610	276.4	18.6
compacted	3,441	61,640	192.2	17.9
Flight Training	780	141,200	4.3	181.0
Shipyard	1,020	20,000	52.0	19.6

Let r denote the ratio of number of suppliers to number of demand points (i.e., r=n/m). Then,

$$H=(2+r+1/r)/f.$$

Since $r+1/r\geqq 2$, it follows that $H\geqq 4/f$. Therefore, a sufficient condition for the list structure to use less space is that f<2/9. It is not uncommon, in practice,[2] for the number of links to be less than 20 percent of the potential number (i.e., f<2), in which case the list structure would require less space, even when we fix h=18.

Now let us consider storage reduction for special networks which arise naturally in practice. The transportation problem exemplifies a bipartite network since the vertex set can be partitioned into two sets (suppliers and demand points), such that every successor of a vertex in V_1 is in V_2, and the vertices in V_2 have no successors. The target-battery network described by Furman and Greenberg[6] is another such case. Both the bit map and list structures may be reduced, as we now describe.

The bit map structure requires only $/V_1/\ /V_2/$ bits, one bit map for each vertex in V_1, each having length $/V_2/$. The list structure still represents /A/ indices, one per arc, but the value of the index size (h) need only be $\lfloor \log_2/V_2/\rfloor$. When the index size is fixed, its value can be less than 15, and we can still represent a large network. For example, if we let h=10 (convenient for CDC 6000's), then V_2 can contain up to 2,047 vertices; the cardinality of V_1 does not constrain h. For the target-battery network, h=8 (convenient for IBM 370's) would permit up to 511 batteries, which is much more than the applications cited in Reference 6.

Define the index threshold for the bipartite network as:

$$H_2=/V_1//V_2/\div/A/.$$

If we continue to assume that only a fraction of the potential arcs will be present, then we note $/A/=f\ /V_1/\ /V_2/$, so

$$H_2=1/f.$$

A sufficient condition for the list structure to take less space (with $h\leqq 10$) is: f<.10; a sufficient condition for the bit map structure to take less space (with $h\geqq 8$) is: f>.125.

We now generalize this reduction. Suppose the vertex set can be partitioned into m sets, $V_1 \cup \ldots \cup V_m$, such that every successor of a vertex in V_k is in V_{k+1}. For k=m we distinguish two cases: $V_{m+1}\equiv$empty set, and $V_{m+1}\equiv V_p$ $(1\leqq p\leqq m)$. The first case is exemplified by the transportation problem. The second case allows "feedback" from V_m to V_p, which we shall illustrate shortly.

The storage requirement for the bit map is reduced to:

$$\sum_{k=1}^{m} /V_k/\ /V_{k+1}/$$

bits, since the bit map for each vertex in V_k has length $/V_{k+1}/$.

The list structure still stores /A/ indices, but we may reduce h as in the case of the bipartite network. Define the associated index threshold:

$$H_m = \sum_{k=1}^{m} /V_k/\ /V_{k+1}/ \div /A/.$$

(Note that H_1 is the original H with feedback.)

Let us consider another problem class which is bipartite with feedback. (This is topologically equivalent to an undirected, bipartite graph.) Suppose we have a collection of elements and a collection of sets. Define a network with one vertex per set, say V_1, and one vertex per element, say V_2. If set i contains element j, then two arcs are defined: one from the i-th vertex in V_1 to the j-th vertex in V_2, and its reverse arc.

The packing/covering/partitioning problem class[7,15] may be represented by this type of network. The feedback is useful for processing since it describes those to which sets an element belongs.

PROCESSING

There are many algorithms to solve various network optimization problems. In this section, we shall consider two fundamental processes and the relative merits of the two structures.

One common process is the interrogation of the successors of a given vertex. This can be expressed as the transaction, "Fetch the list of successors of vertex i." For example, most (node/arc) labeling methods[1,2,3,10,18] use this process.

Clearly, the bit map structure requires unpacking. However, for certain implementations (e.g. CDC 6000's and UNIVAC 1100's), the list structure also requires unpacking, although it is of a different form. The actual timings depend upon problem and machine characteristics, but we can, and shall, develop a representative relative measure for this process. To help fix ideas, COMPASS macros were written to fetch the next index or declare no more successors exist. The loading of base addresses is the same for both structures, but the instructions to complete the fetch depends upon the structure. It is this latter difference which we now consider.

Using four indices per word (i.e., h=15), the list structure will execute an average of 3.75D instructions for a complete interrogation (discounting load overhead common to both structures). More generally, if the length of an addressable unit (e.g., word) is w, then the number of instructions is 3(1+h/w)D.

For the bit map, we used the normalization instruction to avoid sequential testing of each bit. If the length of the bit map is /V/, then the average number of instructions is /V/(3/20)+180D÷/V/. In general, the average number of instructions is 3wD÷/V/+9/V/÷w. For sufficiently large /V/ and D, the *processing ratio* of bit map to list structure, for the successor interrogation, is approximately:

$$T = (9/4)/V/ \div hD.$$

Therefore, we have

$$T = 2.25\ H/h.$$

For fixed h, the relative processing time depends on the index threshold: index size ratio. The bit map structure uses more (less) instructions if this ratio is greater (less) than 4/9. A consequence of this relationship is that the list structure will process with fewer instructions if it takes less space (i.e., h<H).

Table II below lists, for each of the Glover-Klingman[8] problems, the associated thresholds times the computed constant, 2.25. Those problems which favor the list structure spatially must also process with fewer instructions, on the average, for this transaction. Specifically, those problems are: Treasury, Cotton Gin (reduced and compacted), and Shipyard. In fact, the reduced form of the Cotton Gin problem is expected to process at least 34 times faster with the list structure! Its first form requires more space to use the list structure, but it would process faster for h≦16. The Flight Training problem processes with fewer instructions, on the average, using the bit map structure; hence, for that problem, bit maps are more frugal with both space and "time."

Now let us consider another type of transaction which appears to favor the bit map structure due to the natural correspondence between set and logical operations.

For covering problems, it is useful to test whether one set is a subset of another.[7, p. 302-3] The associated constraints may include partitioning (or "strict covering"); then the set difference is desired in case the subset test passes.

Since we can, and shall, assume the degree of each

TABLE II

Problems	Processing Threshold (2.25H)
Treasury	90.0
Cotton Gin	
first form	16.2
reduced	621.9
compacted	432.45
Flight Training	9.7
Shipyard	117.0

vertex is available, the test on a pair of vertices can be branched into a test for equality and a test for proper inclusion, with the potential subset identified. Further, we shall assume that the list structure has each successor list sorted.

It can easily be verified that the coding of the set equality test is essentially independent of the representation and can be tested as array-equality comparison. (It is immaterial whether the elements of the array are pieces of a bit map or several indices; only the number of loadable entries in the array counts.) Thus, the processing ratio for the equality test is:

$$T=/V/\div hD=H/h.$$

This says that the list structure takes less time to process an equality test if, and only if, it takes less space than the bit map structure.

Now let us consider the subset test. The bit map can be processed with 8 COMPASS instructions, which are executed for every word in the bit map. Thus, the number of instructions executed is $8/V/\div w$.

Define $b=h/w$ (=length of a bit map in h units). The list structure, on the average, executes $6D(1+b)$ instructions for fetch indices, and the average number of other instructions needed is 3d, where d is the larger degree. Thus, the list structure uses $9D+6Db+3(d-D)$, on the average.

If we suppose d-D is relatively small (compared to D), then the processing ratio is approximately:

$$T\div\frac{8/V/}{9Dw+6Dh}=BH/h,$$

where

$$B=8b/(6b+9).$$

Again, the processing ratio is proportional to the ratio of index threshold to index size, except now the constant of proportionality (which depends on h) is less than 1. In this case the bit map structure processes with fewer instructions, on the average, if it requires less space. For example, suppose we have $b=1/6$ (i.e. $h=10$ and $w=60$) for a covering problem having m elements and an average of e elements per set. Then,

$$H_2=m/e$$

so

$$T=.0013m/e.$$

Therefore, despite first appearances, there is a threshold whereby the list structure processes faster (and uses less space); that is, the processing ratio is greater than 1 if $m/e>750$. If we have 750 elements, then the average number of elements per set would have to be 1, not very likely. However, if we have 75,000 elements, then we need the average number of elements per set to be only 100. For some (large) models this is within reality.

We may infer from these two fundamental processes that, in general, the influence of density on processing ratio is the index threshold proportionality relation:

$$T=kH/h.$$

When $k<1$, the process favors the bit map structure; when $k>1$, the process favors the list structure. Computation of the constant (k) may be done at the macro level to predict relative performance.

CONVERSION

In this section we shall present algorithms to convert from one representation to another, designed to minimize the extra work space required.

To conserve space we wish to overwrite as much of the bit map as possible, while forming the list structure, but we cannot destroy information until it is no longer needed. Let B be the base address of the bit map (stored contiguously), and let L be the base address of the list structure. Let us suppose, for definiteness, that $L<B$. Then, $B-L$ is the work space to be minimized.

The conversion procedure begins with vertex 1 and unpacks its bit map, starting at B, into the list structure, starting at L. At a general iteration, the bit map for vertex i is unpacked into the list. Its bit map is located between $B+(i-1)b$ and $B+ib$; its list structure is located between $L+d_{i-1}$ and $L+d_i-1$, where $d_i=$ sum of the degrees of the first i vertices. Therefore, to avoid premature overwrite, it is sufficient to require:

$$L+d_i-1\leqq B+(i-1)b \quad \text{for } i=1,\ldots,/V/.$$

Define

$$M=\max[d_i-ib: 1\leqq i\leqq /V/].$$

Then, we can set

$$B=L+b-1+M.$$

To convert from a list structure to a bit map representation, we transform in reverse order. If the bit map structure uses a sufficient amount of more space than the list structure, no other extra space is needed. Specifically, if $b/V/+M+b-1\geqq/A/$, then this conversion can be accomplished within the total region (L to $L+b/V/+M+b-1$).

REFERENCES AND BIBLIOGRAPHY

1. Bellmore, M., H. J. Greenberg and J. J. Jarvis, "Multi-Commodity Disconnecting Sets," *Mgt. Sci.*, 16, 6, 1970, pp. 427-433.
2. Barr, R. S., F. Glover and D. Klingman, "An Improved Version of the Out-of-Kilter Method and a Comparative Study of Computer Codes," *Math. Prog.*, 7, 1, 1974, pp. 60-87.
3. Berztiss, A. T., *Data Structures Theory and Practice*, Academic Press, 1971.
4. Busacker, R. G. and T. L. Saaty, *Finite Graphs and Networks: An Introduction with Applications*, McGraw-Hill, 1965.
5. Ford, L. R. and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, 1962.
6. Furman, G. G. and H. J. Greenberg, "Optimal Weapon Allocation with Overlapping Area Defenses," *Opns. Res.*, 21, 6, 1973, pp. 1291-1308.

7. Garfinkel, R. S. and G. L. Nemhauser, *Integer Programming*, Wiley, 1972.
8. Glover, F. and D. Klingman, "Network Applications in Industry and Government," MSRS 75-20, University of Colorado, 1975.
9. ——— and ———, "Capsule View of Future Developments on Large Scale Network and Network-Related Problems," Research Rept. CCS 238, University of Texas, 1975.
10. Hellerman, E. and D. C. Rarick, "The Partitioned Preassigned Pivot Procedure (P4)," in *Sparse Matrices and Their Applications*, D. J. Rose and R. A. Willoughby (eds.), Plenum Press, 1972, pp. 65-76.
11. Kalan, J. E., "Aspects of Large-Scale In-Core Linear Programming," *Proceedings of ACM*, 1971, pp. 304-313.
12. Knuth, D., *The Art of Computer Programming, volume 1: Fundamental Algorithms*, Addison-Wesley, 1968 (rev., 1973).
13. Lefkovitz, D., "The Large Data Base File Structure Dilemma," *Journ. of Chem. Inf. & Comp. Sci.*, 15, 1975, pp. 14-19.
14. MacVeigh, D. T., *Effect of Data Representation on Efficiency of Sparse Matrix Operations*, M.S. Thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 1975.
15. Mulvey, J. M., "A Network Relaxation Approach for the Set Partitioning and Set Covering Models," HBS 75-37, Harvard University, 1975.
16. Orchard-Hays, W, *Advanced Linear Programming Computing Techniques*, McGraw-Hill, 1968.
17. Pooch, U. W. and A. Nieder, "A Survey of Indexing Techniques for Sparse Matrices," *Comp. Surv.*, 5, 1973, pp. 109-133.
18. Simonnard, M., *Linear Programming*, Prentice-Hall, 1966.

A practitioner's guide to the state of large scale network and network-related problems

by FRED GLOVER
University of Colorado
Boulder, Colorado
and
DARWIN KLINGMAN
University of Texas
Austin, Texas

ABSTRACT

The primary purpose of this paper is to identify recent major accomplishments and prophesy future trends in the solution, modeling, and human engineering aspects of network and network-related problems.

Network and network-related problems include such mathematical problems as assignment, transportation, transshipment, multi-commodity networks, generalized networks, plant location problems, fixed charge networks, and constrained networks. In fact, it has recently been shown that the 0-1 integer programming problems are equivalent to 0-1 generalized network problems.

Network-related problems that arise "naturally" (i.e., those whose formulations make conspicuous reference to a network structure) cover a diverse spectrum of practical applications. These applications include problems in telecommunication, microdata set merging, cash flow, multi-national currency exchange, manpower planning, waste disposal, water resource management, aircraft refueling, student-professor-classroom scheduling, modular design, cutting stock, machine loading, aircraft scheduling, off-shore oil well location and drilling, and production and distribution scheduling, to mention only a few.

SETTING THE STAGE

The primary purpose of this paper is to identify recent accomplishments in formulating and solving network and network-related problems, and to propose an integrated sequence of future developments to take advantage of the rich possibilities for new advances and applications in the field. To achieve these purposes with reasonable conciseness, the next section presents a *very brief* overview of the recent major developments in network and network-related problems. These developments are then partitioned and discussed in the following sections.

RECENT HAPPENINGS

The published research on network and network-related problems prior to 1969 primarily focused on characterizing solution algorithms from a mathematical viewpoint. Very little was done in the way of investigating how broad algorithmic principles should be organized and interrelated to provide efficient computer implementation. Additionally no intensive computer code developments or rigorous empirical studies were presented. Basically the determination of the most effective algorithmic principles, and the development of special techniques for exploiting these principles in a highly efficient manner, were neglected areas.

Things have really changed since 1969! The following developments have all occurred subsequent to this date:

(1) The first reported transportation and transshipment computer codes based on the dual simplex method.[10,12,20]
(2) the first reported techniques for streamlining the out-of-kilter algorithm to enhance its computer implementation.[1,4]
(3) a number of special updating and labeling techniques and their underlying list structures for improved computer implementation of network algorithms.[5,11,13,17,19,22,28,32]
(4) the first reported transshipment code based on the primal simplex method.[10]
(5) the first transportation and transshipment codes[2,23] designed for solving large-scale problems within reasonable solution times.
(6) major advances in primal simplex network computer codes for generalized network problems[14,26,27] (in our terminology, the "generalized" network problem is the "flow with gains and losses" problem).
(7) the first constrained network computer code.[28]

These code development efforts have been accompanied by rigorous empirical studies to determine the best solution and implementation procedures. Additionally, the past six years have witnessed the formulation and solution of substantially increased numbers of real world problems as network or network-related problems.[6,9,21,25] It appears that many researchers are adopting real world implementation as one of the major goals of their efforts, and this is beginning to serve as a driving force in the development of efficient solution and modeling approaches. Indeed, a number of new modeling techniques have appeared which have substantially broadened the class of network-related problems,[9,24,25] including the discovery[18] that an 0-1 linear program can be modeled as a 0-1 generalized network problem or as a "O-U" transshipment problem. Given these advances it is appropriate and necessary to ponder and reflect on a number of questions. For example: What are the implications of these advances for identifying worthwhile directions for future research? What realms of applications have implicitly been opened up that have not yet been fully recognized? What thresholds are about to be crossed, and of these, which are the most important to examine next? What types of interactions are likely to occur between researchers and practitioners due to the unprecedented practical value of many of these advances? What supplementary developments for enhancing the use (and usefulness) of these advances are likely to emerge? The remaining sections of this paper contain some of our more concrete reflections on these questions.

TRANSPORTATION AND TRANSSHIPMENT PROBLEMS

This section attempts to summarize recent advances in the solution of transportation and transshipment problems. The total work that has been done in this area since 1969 consists of at least 10 man years of work by mathematical programmers and systems analysts. For a more complete discussion of events prior to 1975 the reader should see Reference 6.

The recent developments of simplex computer codes for networks were initiated in two major studies: those of Srinivasan and Thompson[31] and of Glover, Karney, Klingman, and Napier.[12] Both studies introduced a variety of refinements and explored techniques for taking advantage of computational trade-offs that had been left unexamined in the literature to that time. One of the significant findings to emerge from these studies was that the special updating and labeling techniques of the API method[11] (which made use of Ellis Johnson's triple label list representation[22]) contributed markedly to the efficiency of the procedures of Reference 12, and when subsequently incorporated into the methods of Reference 31, improved their performance by a factor of 2.5. This validated the expectation that a critical factor in the development of any network computer code is the manner in which an effective list structure is integrated with the operations of storing and updating the required information.

Having decided that the API method was the best updating technique then available, substantial empirical testing was conducted to determine the best starting and pivoting rules to use with simplex based codes.[10,12,31] Following this testing the first primal and dual simplex based transshipment codes were developed by Glover, Karney, and Klingman[10] and similar testing was conducted. These investigations constituted what could be called the "First Generation" of Modern Transportation and Transshipment Codes (1969-72). This period significantly advanced the speed of solving such problems and saw the first intensive application of the combined skills of mathematical programmers and systems analysts in the network area. These investigations uncovered a number of fallacies in the accepted folklore of that time. For instance, it was shown[10,12] that the new simplex based transportation and transshipment codes were at least an order of magnitude (i.e., ten times) faster than the best available out-of-kilter and dual simplex codes, and at least two orders of magnitude (i.e., a hundred times) faster than state-of-the-art commercial LP systems. Thus, considerable doubt was cast on the vintage preconception that the out-of-kilter method is inherently superior to a specialized primal simplex method, and on the more recent notion advanced by commercial LP firms that special purpose network codes are not significantly faster than state-of-the-art LP systems.

The First Generation of network advances also included the first major streamlining of the out-of-kilter algorithm for computer implementation by Barr, Glover, and Klingman.[4] This produced a sixfold improvement in solution times over previous out-of-kilter codes. Using these improvements the out-of-kilter approach was still unable to overtake the efficiency of the best simplex-based approach either in solution speed or in memory requirements.

Following in the aftermath of all these developments, and taking advantage of them, the Second Generation may be conceived as originating in 1973 with the development of a new type of list structure and updating scheme for storing and manipulating the spanning tree basis. This new approach by Glover, Klingman and Stutz,[17] called the Augmented Threaded Index (ATI) method, provided both increased efficiency and reduced memory requirement over the previously best API method. Mulvey[29] elaborated on this development by integrating the ATI approach with the distance function concept of Srinivasan and Thompson.[32] Mulvey's integration of the distance function and the ATI approach improved solution times another 10 percent at the expense of an additional node-length array of computer storage. One significant contribution of the ATI method stems from the fact that it became possible to design an in-core out-of-core simplex transportation code which only uses four node length arrays to maintain and update all basis information. This advance is of paramount importance for solving the extremely large problems that sometimes arise in practical applications. One of these is a micro-data set merger problem recently formulated by the U.S. Treasury. This problem is a transportation problem which contains 50,000 nodes and 62 million arcs. The U.S. Treasury requires problems from this class to be solved several times a year on a UNIVAC 1108, and contracted Analysis, Research, and Computation (ARC), Inc. to design and implement a code capable of meeting these requirements. By using the ATI labeling and updating method, such a code[2] has been developed in FORTRAN and to date has run problems with 5000 nodes and 625,000 arcs in less than four minutes of CP time (and nine minutes total processing time) on a UNIVAC 1108.

The most significant finding of this "Second Generation" in the area of pivot selection strategies was made by Mulvey,[29] who showed that solution times can be substantially reduced for problems with more than a thousand nodes by using a special form of a "candidate list" pivot strategy.

During the early part of this Second Generation, McBride and Graves[28] developed a general technique called the factorization method. The first implementation of this method in the network setting demonstrated the possibility of specialized application in this area. The implementation, however, proved to be substantially slower than the API and ATI procedures and required even more computer memory than the initial implementation of the API approach. These limitations were of course due in large part to the experimental nature and lack of refinement of the first implementation of any approach. Indeed, a later improved implementation by Graves (not reported in the literature) showed that the method was susceptible to substantial improvement, although it remained twice as slow as the implementation of the ATI procedure reported in Reference 10 and continued to require more computer memory, when both procedures were tested on the same computer and same problems at General Motors, Inc. Nevertheless, the factorization ideas contain a good deal of ingenuity, and future implementations may be found that are superior to those developed so far.

This concentrated burst of activity that we have called the Second Generation was quite short, and in referring to it we have indeed included mention of developments that properly carry over to the present "Third Generation." We choose to demark the beginning of the Third Generation as coinciding with the development of a third updating and labeling scheme for improving the implementation of network algorithms. On the basis of sound analytical arguments, it is quite likely that all updating and labeling schemes for simplex network codes will henceforth only augment the information kept by this approach. This new approach by Glover and Klingman[19] which appeared in mid-1974, augments the ATI method with two new functions, called the "cardinality" and "last node" functions. The use of these functions in conjunction with the ATI method makes it possible to update flows and node potentials at each iteration with a marked increase in efficiency. Further, both of these functions together require less work to update than the distance function. As a final bonus, the "cardinality function" can accommodate all of the relevant functions filled by the distance function, and hence can replace it. The net result of all these advantages produces a substantially improved procedure for implementing basis exchange operations. These latest developments are currently being tested and are expected to provide another twofold improvement in solution speed.

The current work of Aashtiani and Magnanti[1] on improved out-of-kilter procedures bears watching. In addition, Shapiro[30] has noted a previously overlooked distinction between the out-of-kilter method and the primal-dual method that may hold promise for exploiting the latter method by subgradient optimization and by the techniques of Reference 19. Thus the Third Generation promises to produce several enhancements in the implementation of the primal simplex and out-of-kilter approaches and may initiate a new generation of hybrid solution approaches.

FUTURE DEVELOPMENTS FOR TRANSPORTATION AND TRANSSHIPMENT PROBLEMS

In spite of the notable gains in network solution techniques since 1969, there is still a major area that remains to be explored. This is the area of designing appropriate techniques for accommodating and taking advantage of degeneracy. Computational testing has shown that approximately 90 percent of the pivots for transshipment problems with more than 1000 nodes are degenerate. Presently, schemes are being computationally tested by Elam, Klingman and Stutz to respond to this situation. The techniques being tested are designed to circumvent pivots and to make them judiciously. It is estimated that an effective scheme for handling degeneracy could easily reduce solution times by another 50 percent.

Another development which should prove to be a significant future advance in mathematical programming is the implementation of network algorithms on mini-computers. Given the dramatic recent improvements in the speed and memory requirements of network methods, and the major advances in mini-computer design, it now appears possible to implement relatively efficient in-core out-of-core transportation and transshipment codes in an assembly level language on most mini-computers. This belief is partly based on the work reported in References 2 and 23, which indicates that simplex based codes do not suffer undue increases in solution times by keeping the problem data in external computer memory. Further the FORTRAN code[10] has been tested on the General Dynamic's Nova computer. Preliminary testing showed that problems with 200 nodes and 600 arcs could be solved in less than a minute on the Nova. The emergence of network codes on mini-computers would afford the potential to substantially increase the use of mathematical programming in the real world. For example, such a development would make it possible to demonstrate OR techniques easily and conveniently to managers in their own offices, at management seminars, and in other countries. Further, since the computer could be dedicated to network applications, it would be feasible and highly desirable to design the operating system to minimize the human engineering aspects of problem solving—that is, to minimize the difficulties of entering, modifying, and verifying problem data, passing the problem data to the solution code, interpreting the output, and so forth. With a dedicated system such problems could be almost nonexistent to the user.

TRANSFER OF TRANSPORTATION AND TRANSSHIPMENT TECHNOLOGY TO RELATED PROBLEMS

Assignment problems

With the exception of the work by Rao and Fong, very little research has been directed toward specializing the recent network innovations to the solution of assignment problems. Consequently the most efficient way to solve assignment problems is currently unknown. Within the next two years this question will be resolved.

Current research by Barr, Glover and Klingman (to appear) has resulted in development of a new extreme point algorithm that computationally overcomes the "massive degeneracy" that is characteristic of assignment problems and drastically reduces computer memory requirements for this class of problems.

Generalized networks

Currently the updating and labeling techniques for transportation and transshipment problems are being extended to generalized network problems also called "weighted networks" or "networks with gains." The fundamental relationships by which such extensions can be carried out effectively are developed in Glover, Klingman and Stutz,[13] which shows how the diverse basis updating configurations (of which there are more than a dozen separate cases) can be consolidated into a single "general case." This result builds upon the early work of Balas and Hammker[3] who characterized the basis structure for this class of problems as a forest of "1-trees" (i.e., trees augmented by additional arcs). Additional computational simplifications of the algorithmic steps for generalized networks have appeared. For example, Maurras[27] and Glover and Klingman[15] indicate how to simplify basis updating calculations by characterizing an appropriate sequence for tracing out paths in the one-trees and for proceeding through all nodes in a subtree of a one-tree. These events led to the development of efficient simplex-based codes for generalized networks by Glover, Klingman, and Stutz,[14] Langley,[26] and Maurras.[27] Present computational results indicate that generalized network problems require about four times longer to solve than transportation and transshipment problems of similar dimension. However, a series of algorithmic and implementation advances in the next several years should substantially reduce the already impressive solution times of such problems.

Constrained network

By a constrained network we mean an assignment, transportation or transshipment problem which also includes extra linear constraints. Klingman and Russell[24] have developed a special basis compactification procedure for this class of problems that maintains all of the network as the "implicit" portion of the basis. The operations involving the implicit portion of the basis are simply carried out as they would ordinarily be for the underlying network. Thus it is possible to take advantage of the efficient procedures previously described for solving network problems. Preliminary computational implementations by Glover, Karney, Klingman, and Russell[16] indicate that this procedure is highly efficient for solving problems with one or two extra constraints. Such problems typically required only twice as long to solve as the underlying network problem. A prototype code using related ideas to handle more than two extra constraints has recently been developed by Chen and Saigal[7] although without attempting to incorporate the essential refinements required to take full advantage of the network structure or to optimize the computer memory requirements. We strongly anticipate that refined codes to handle larger numbers of extra constraints will be developed in the near future. The results of Reference 16 suggest that these codes will be much more efficient than general linear programming codes for solving this class of problems.

The importance of the basis compactification area for solving constrained network problems is difficult to overrate. It will probably be the focus of numerous active investigations and will produce the next breakthrough, similar to the GUB breakthrough, in the developments of linear programming codes. This belief rests on the observations that

(a) GUB is a specialization of the network basis compactification procedures of Reference 24,
(b) the network basis compactification extends GUB in the sense that it eliminates the nonoverlapping variable requirement of GUB.
(c) the number of real world LP problems which have network substructures is quite large.

Modeling

As a result of the projected algorithmic and implementation activities for generalized and constrained network problems, we foresee a flurry of modeling advances for these problems. In our opinion, such problems have a vastly richer domain of application than ordinary networks; e.g., examples include cash flow problems, multi-commodity problems, waste disposal problems, water resource management problems, plastic problems of civil engineering, production scheduling problems, machine loading problems, goal programming networks, stochastic and constrained regression problems, and multi-national currency exchange problems, to mention only a few.

An important educational function that should soon be adopted by members of the Operations Research and Management Science fields is to train and guide modelers and decision makers in visualizing their problems within network-related frameworks—particularly those of generalized network or constrained network problems when this is possible. A trend is already underway[18,21] toward identifying problems that can be framed cost-effectively as network-related problems. Network-related problems provide natural formulations for a vastly larger number of real world problems than previously suspected. At the same time, researchers are discovering that a variety of other real world problems can be easily transformed into this format.[18,21] Such models and formulation possibilities have been virtually unknown in the literature before now because of the lack of efficient ways to solve such problems as a class distinct from general LP models.

It is further believed that the effort to help people think of their problems in a network-related format will have far reaching and beneficial consequences in two ways. First, this type of visualization will enable decision-makers to formulate their problems initially by means of drawings and diagrams. This is much easier for the nontechnical person than trying to formulate his problems in terms of equations. The result will be to provide increased recognition of the pervasiveness and significance of these types of problems, and thus to enlarge the number of important real world problems that are being solved by OR methodology. The second and probably more significant consequence will derive from the fact that these types of problems can often be solved with dramatic success, even in large-scale applications. Consequently, a much greater appreciation of OR by industry and government will come about, permitting the widespread disillusionment that followed on the heels of many attempted LP and IP implementations to be reversed. Indeed, it seems conceivable that such developments may spearhead a new liaison between OR and its users. We have personally had first hand experience of this upsurge of interest by industry and government when such modeling and solution approaches are used. Recent real world applications which have been successfully solved via these modeling and solution approaches are summarized in Reference 21.

REFERENCES

1. Aashtiani, H. and T. L. Magnanti, "Solving Large Scale Network Optimization Problems," Presented at ORSA/TIMS conference, Chicago, April 1975.
2. Analysis, Research, and Computation, Inc., "Development and Computational Testing on Large Scale Primal Simplex Network Codes," ARC Technical Research Report, P. O. Box 4067, Austin, TX 78765, 1974.
3. Balas, E. and P. L. Ivanescu (Hammker), "On the Generalized Transportation Problem," *Man. Sci.* 11, pp. 188-202, 1964.
4. Barr, R. S., F. Glover and D. Klingman, "An Improved Version of the Out-of-Kilter Method and a Comparative Study of Computer Codes," *Mathematical Programming,* 7, 1, pp. 60-87, 1974.
5. Bradley, G., G. Brown and G. Graves, "A Comparison of Storage Structure for Primal Network Codes," Presented at ORSA/TIMS conference, Chicago, April 1975.
6. Charnes, A., F. Glover, D. Karney, D. Klingman and J. Stutz, "Past, Present, and Future of Development, Computational Efficiency, and Practical Use of Large Scale Transportation and Transshipment Computer Codes," *Computers and Operations Research,* 2, pp. 71-81, 1975.
7. Chen and Saigal, "A Primal Algorithm for Solving a Capacitated Network Flow Problem with Additional Linear Constraints," Presented at ORSA/TIMS conference, Chicago, April 1975.
8. Elam, J., D. Klingman and J. Stutz, "Degeneracy and its Computational Resolution," forthcoming.
9. Gavish and P. Schweitzer, "An Algorithm for Combining Truck Trips," *Trans. Sci.,* 8, 1, pp. 13-24, 1974.
10. Glover, F., D. Karney and D. Klingman, "Implementation and Computational Study on Start Procedures and Basic Change Criteria for a Primal Network Code," *Networks,* 4, 3, pp. 191-212, 1974.
11. Glover, F., D. Karney and D. Klingman, "Augmented Predecessor Index Method for Location Stepping Stone Paths and Assigning Dual Prices in Distribution Problems," *Trans. Sci.,* 6, 1, pp. 171-181, 1972.
12. Glover, F., D. Karney, D. Klingman and A. Napier, "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," *Man. Sci.,* 20, 5, pp. 793-819, 1974.
13. Glover, F., D. Klingman, and J. Stutz, "Extensions of the Augmented Predecessor Index Method to Generalized Network Problems," *Trans. Sci.,* 7, 4, pp. 377-384, 1973.
14. Glover, F., D. Klingman and J. Stutz, "Implementation and Computational Study of a Generalized Network Code," 44th National Meeting of ORSA, San Diego, California, 1973.
15. Glover, F. and D. Klingman, "A Note on Computational Simplifications in Solving Generalized Transportation Problems," *Trans. Sci.,* 7, pp. 351-361, 1973.

16. Glover, F., D. Karney, D. Klingman and R. Russell, "Solving Singularly Constrained Transshipment Problems," Research Report CCS 212, Center for Cybernetic Studies, University of Texas, Austin, Texas, 1974.
17. Glover, F., D. Klingman and J. Stutz, "Augmented Threaded Index Method for Network Optimization," *INFOR*, 12, 3, pp. 293-298, 1974.
18. Glover, F. and J. Mulvey, "Equivalence of the Zero-One Integer Program to Discrete Generalized and Pure Networks," MSRS 75-19, University of Colorado, 1975.
19. Glover, F. and D. Klingman, "Improved Labeling of L. P. Bases in Networks," Research Report CS 218, Center for Cybernetic Studies, University of Texas, Austin, Texas, 1974.
20. Glover, F. and D. Klingman, "Double-pricing Dual and Feasible Start Algorithms for the Capacitated Transportation (distribution) Problems," University of Texas, Austin, Texas, 1970.
21. Glover, F. and D. Klingman, "Network Applications in Industry and Government," MSRS 75-20, University of Colorado, May 1975.
22. Johnson, E., "Networks and Basic Solutions," *Opns. Res.* 14, pp. 89-95, 1966.
23. Karney, D. and D. Klingman, "Implementation and Computational Study on In-Core Out-of-Core Primal Network Code" to appear in *Opns. Res.*
24. Klingman, D. and R. Russell, "On Solving Constrained Transportation Problems," *Opns. Res.* 1, pp. 91-107, 1975.
25. Klingman, D., P. Randolph, and S. Fuller, "A Cotton-Pickin Cotton Ginning Problem," to appear in *Opns. Res.*
26. Langley, R. W., "Continuous and Integer Generalized Flow Problems," Ph.D. Dissertation, Georgia Tech., 1973.
27. Maurras, J. F., "Optimization of the Flow Through Networks with Gains," *Mathematical Programming*, 4, 2, pp. 135-145, 1972.
28. McBride, R. D. and G. W. Graves, "The Use of Inherent Triangularity in Network Problems," presented at 44th National Meeting of ORSA, San Diego, November 1973.
29. Mulvey, J., "Column Weighting Factors and Other Enhancements to the Augmented Threaded Index Method for Network Optimization," Joint ORSA/TIMS Conference, San Juan, Puerto Rico, 1974.
30. Shapiro, J. F., "A Note on the Primal-Dual and Out-of-Kilter Algorithms for Networks Optimization Problems," MIT, March 1975.
31. Srinivasan, V. and G. L. Thompson, "Benefit-Cost Analysis of Coding Techniques for the Primal Transportation Algorithm," *JACM*, 20, pp. 194-213, 1973.
32. Srinivasan, V. and G. L. Thompson, "Accelerated Algorithms for Labeling and Relabeling of Trees with Application for Distribution Problems," *JACM*, 19, 4, pp. 712-726, 1972.

Low-cost residue number systems for computer arithmetic

by BEHROOZ PARHAMI
Arya-Mehr University of Technology
Tehran, Iran

ABSTRACT

The representation of integers by their residues with respect to a set of pairwise-prime moduli is known as the residue number representation system and has been shown to have several advantages over conventional number systems for digital computers. In this paper, residue systems are considered for which each modulus is of the form 2^b-1. Such systems result in relatively high storage efficiency as well as simple algorithms for addition, subtraction multiplication, conversion, and reconversion; hence the name "low-cost." The question of existence for low-cost residue number systems is examined. It is shown that the additional storage requirement with respect to binary representation is at most one bit per word. Guidelines are given for optimal selection of the set of moduli to represent a desired range of integers. Algorithms for various operations in a low-cost residue system are described.

INTRODUCTION

When dealing with large numbers in digital computers, the computations are slowed down because of the requirement for carry or borrow propagation through many stages of logic in addition and subtraction operations and for long iterative algorithms to perform multiplication and division. Attempts to eliminate the propagation of carries and borrows have resulted in proposals for stored-carry[1] and signed-digit[2] number representation systems. The residue number system[3] does not totally eliminate carry propagation but limits it to within a few stages by representing large numbers as an ordered set of smaller numbers that can be processed independently and in parallel. This is particularly advantageous in multiplication which becomes almost as simple and as fast as addition. However, the complexity of division in residue number systems makes them unsuitable for general-purpose use.

In this paper, residue number systems are reviewed briefly and their properties are enumerated. A class of residue number representation systems which results in relatively high storage efficiency as well as simple algorithms for addition, subtraction, multiplication, conversion, and reconversion algorithms is introduced. The questions of existence, selection, storage efficiency, and algorithms for such "low-cost" residue systems are examined. The storage requirement for each word is shown to be within one bit of the binary representation. Algorithms needed for basic operations and conversions are discussed.

RESIDUE NUMBER SYSTEMS

A *residue number system*[4,5] is one in which a numerical value n is represented by a k-tuple whose components are the residues of n with respect to an ordered set of k moduli

$$p = \langle p_1, p_2, \ldots, p_k \rangle \tag{1}$$

which are relatively prime pairwise. Hence, n is represented by the k-tuple

$$r = \langle r_1, r_2, \ldots, r_k \rangle \tag{2}$$

such that

$$r_i = p_i | n \; ; \; i = 1,2, \ldots, k \tag{3}$$

where $\rho = \mu | \xi$ means that ρ is the smallest non-negative integer satisfying $\xi = \rho + \beta\mu$ for some integer β. The *range* of a residue system (i.e., the number of distinct values representable) is:

$$N = \prod_{i=1}^{k} p_i \tag{4}$$

To represent a negative integer $-n$, we simply represent the positive integer $N-n$ since we have

$$p_i | (N-n) = p_i | (-n) \; ; \; i = 1,2, \ldots, k \tag{5}$$

The integer $N-n$ is the *additive inverse* of n and is denoted by $\bar{n}$. The residue representation $\bar{r}$ of $\bar{n}$ has the following relation with the representation r of n:

$$\bar{r}_i = p_i | (p_i - r_i) \; ; \; i = 1,2, \ldots, k \tag{6}$$

If binary representation is used for the residues, the number of bits required for storing each value in the residue system is

$$B = \sum_{i=1}^{k} b_i = \sum_{i=1}^{k} \lceil \log_2 p_i \rceil \tag{7}$$

which is always greater than or equal to $\lceil \log_2 N \rceil$, the number of bits needed for the binary representation of N distinct values. Hence, a residue number system is less efficient than the binary representation in terms of storage space.

Addition and multiplication in a residue system are done by performing the corresponding operation (modulo p_i) on the i-th residues of the two numbers, independently of other residues. Hence, showing the sum and product of $x = <x_1, x_2, \ldots, x_k>$ and $y = <y_1, y_2, \ldots, y_k>$ by s and m, respectively, we can write:

$$s_i = p_i | (x_i + y_i)\ ;\ i = 1,2, \ldots, k \qquad (8)$$

$$m_i = p_i | (x_i \cdot y_i)\ ;\ i = 1,2, \ldots, k \qquad (9)$$

Subtraction is performed by adding the additive inverse, as defined by (6). Thus, the carry propagation delay for addition and subtraction is reduced and the construction of very fast multiplication circuits is made possible. However, comparison of magnitudes, and hence division, and also detection of overflow conditions are fairly complex in residue number systems. Hence, such systems are not suitable for general-purpose use.

To find the normal representation n of a residue number $r = <r_1, r_2, \ldots, r_k>$, the following equation may be used

$$n = N | \sum_{i=1}^{k} \left((r_i c_i \frac{N}{p_i} \right) \qquad (10)$$

where the coefficient c_i is selected to be the smallest integer satisfying

$$c_i = p_i (1 + \beta_i p_i) / N \qquad (11)$$

for some integer β_i.

Another reconversion process uses the following algorithm which is a formalization of the procedure given in Reference 5.

Algorithm R (12)

```
[1] $v \leftarrow r; n \leftarrow 0; w \leftarrow 1; u_j \leftarrow 1, j = 1,2, \ldots, k; i \leftarrow k$
[2] Find smallest integer d such that for some $\beta$,
                                   $d = (v_i + \beta p_i) / u_i$
[3] $n \leftarrow n + wd$
[4] For $j = 1,2, \ldots, k$ set $v_j \leftarrow p_j | (v_j - u_j d)$ and
                                   $u_j \leftarrow p_j | (u_j p_i)$
[5] $w \leftarrow w p_i$
[6] $i \leftarrow i - 1$
[7] if $i > 0$ then go to Step [2] else stop
```

LOW-COST RESIDUE SYSTEMS

A residue number representation system is *low-cost* if each modulus p_i is selected such that:

$$p_i = 2^{b_i} - 1;\ b_i > 2 \qquad (13)$$

The name "low-cost" is justified because of the relatively high storage efficiency and the simplicity of addition subtraction, multiplication, conversion, and reconversion algorithms as will be seen in the remainder of this paper. In this section, we will only concentrate on the existence of such systems and their storage efficiency.

The selection of b_i's must be made such that the resulting p_i's are pairwise prime. It can be proven that p_i and p_j are relatively prime if and only if the corresponding b_i and b_j are relatively prime (see Theorem 1 in the Appendix). Using this result, Table I has been constructed to show the maximal sets of pairwise-prime b_i's for $b_i \leqq 20$, since making b_i larger than 20 may defeat the advantage of residue number systems in breaking long numbers into several short components. We define, as a measure of this advantage, the *dissection factor:*

$$\delta = \max_i (b_i) / \Sigma_i b_i \qquad (14)$$

Table II shows possible selections of b_i's for a given total number of bits, B, which satisfy the following criteria, in the order given: (1) Minimum value of δ, and (2) Smallest number of b_i's. The second criterion is justified by the fact that once the size of the longest group is fixed at its minimum value, no speed advantage results from making the other groups shorter. Figure 1 shows the same results graphically.

We define as a measure of storage efficiency, the ratio of N to the range of the binary system with the same number of bits:

$$n = N/2^B = \prod_{i=1}^{k} (p_i / 2^{b_i}) = \prod_{i=1}^{k} (1 - 2^{-b_i}) \qquad (15)$$

It can be proven (see Theorem 2 in the Appendix) that

TABLE I—Maximal Compatible Sets of b_i's ($b_i \leq 20$) for Low-Cost Residue Number Systems

Set No.	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
1	2	3	5	7	11	13	17	19
2	2	5	7	9	11	13	17	19
3	2	7	11	13	15	17	19	
4	3	4	5	7	11	13	17	19
5	3	5	7	8	11	13	17	19
6	3	5	7	11	13	16	17	19
7	3	5	11	13	14	17	19	
8	3	7	10	11	13	17	19	
9	3	7	11	13	17	19	20	
10	4	5	7	9	11	13	17	19
11	4	7	11	13	15	17	19	
12	5	6	7	11	13	17	19	
13	5	7	8	9	11	13	17	19
14	5	7	9	11	13	16	17	19
15	5	7	11	12	13	17	19	
16	5	7	11	13	17	18	19	
17	5	9	11	13	14	17	19	
18	7	8	11	13	15	17	19	
19	7	9	10	11	13	17	19	
20	7	9	11	13	17	19	20	
21	7	11	13	15	16	17	19	
22	11	13	14	15	17	19		

TABLE II—Best Choices for b_i's in a Low-Cost Residue Number System with a Given Total Number of Bits (B)

B	b_1	b_2	b_3	b_4	b_5	b_6	δ(%)
5	2	3					60.0
7	3	4					57.1
8	3	5					62.5
9	4	5					55.6
10	2	3	5				50.0
11	5	6					54.5
12	3	4	5				41.7
13	6	7					53.8
14	2	5	7				50.0
	3	4	7				
15	3	5	7				46.7
16	4	5	7				43.8
17	2	3	5	7			41.2
18	5	6	7				38.9
19	3	4	5	7			36.8
20	5	7	8				40.0
21	5	7	9				42.9
22	5	8	9				40.9
23	3	5	7	8			34.8
24	7	8	9				37.5
25	4	5	7	9			36.0
26	7	9	10				38.5
27	7	9	11				40.7
28	7	10	11				39.3
	8	9	11				
29	5	7	8	9			31.0
30	9	10	11				36.7
31	3	7	10	11			35.5
	4	7	9	11			
	5	7	8	11			
32	5	7	9	11			34.4
33	5	8	9	11			33.3
34	2	5	7	9	11		32.4
	3	5	7	8	11		
35	7	8	9	11			31.4
36	4	5	7	9	11		30.6
37	7	9	10	11			29.7
38	5	9	11	13			34.2
39	7	8	11	13			33.3
	7	9	10	13			
40	5	7	8	9	11		27.5
41	5	11	12	13			31.7
	7	10	11	13			
	8	9	11	13			
42	2	7	9	11	13		31.0
	3	7	8	11	13		
	4	5	9	11	13		
	5	6	7	11	13		
	5	7	8	9	13		
43	7	11	12	13			30.2
	9	10	11	13			
44	3	7	10	11	13		29.5
	4	7	9	11	13		
	5	7	8	11	13		
45	5	7	9	11	13		28.9
46	5	8	9	11	13		28.3
47	2	5	7	9	11	13	27.7
	3	5	7	8	11	13	
48	5	7	11	12	13		27.1
	7	8	9	11	13		
49	4	5	7	9	11	13	26.6
50	7	9	10	11	13		26.0
51	7	13	15	16			31.4
52	5	9	11	13	14		26.9
53	5	7	8	9	11	13	24.5
54	7	8	11	13	15		27.8
55	11	13	15	16			29.1
56	7	9	11	13	16		28.6
57	11	13	16	17			29.8
	11	14	15	17			
58	2	11	13	15	17		29.3
	3	11	13	14	17		
	5	7	13	16	17		
	5	9	11	16	17		
	5	9	13	14	17		
	5	11	12	13	17		
	7	8	11	15	17		
	7	10	11	13	17		
	8	9	11	13	17		
59	11	15	16	17			28.8
	13	14	15	17			
60	3	11	13	16	17		28.3
	4	11	13	15	17		
	5	9	13	16	17		
	5	11	13	14	17		
	7	8	13	15	17		
	7	9	11	16	17		
	7	11	12	13	17		
	9	10	11	13	17		
61	5	7	9	11	13	16	26.2
62	7	11	13	15	16		25.8
63	7	11	13	15	17		27.0
64	7	11	13	16	17		26.6
	8	11	13	15	17		
	9	11	13	14	17		
65	2	7	11	13	15	17	26.2
	3	5	11	13	16	17	
	5	7	9	11	16	17	
	5	7	11	12	13	17	
	7	8	9	11	13	17	

for any low-cost residue number system $0.5<\eta<1$ from which we can conclude

$$2^{B-1}<N<2^{B} \qquad (16)$$

This shows that the storage requirement for a low-cost residue system is within one bit of the most efficient representation. It also shows that N is an increasing function of B.

To select a low-cost residue system, B must be determined first. To do this, we first note that among all choices for the set of moduli for each value of B, given by Table II, the one for which $\min_i(b_i)$ is a maximum results in the largest possible range (see Theorem 3 in the Appendix). If more than one set has this maximum value for $\min_i(b_i)$, we look at the second smallest b_i in the sets, etc. Table III gives the maximum range obtainable for each value of B. Since, in a low-cost residue system, the storage requirement is dictated by B and the processing speed by $\max_i(b_i)$, the final choice for B among the values which provide adequate range may involve a tradeoff between these two factors. For example if B=51 is sufficient for some desired range, B=52 and B=53 must also be considered for the final selection, since they provide higher processing speeds at the expense of more storage space.

LOW-COST ALGORITHMS

We first note that in dealing with numbers represented in a residue system, the following operations in-

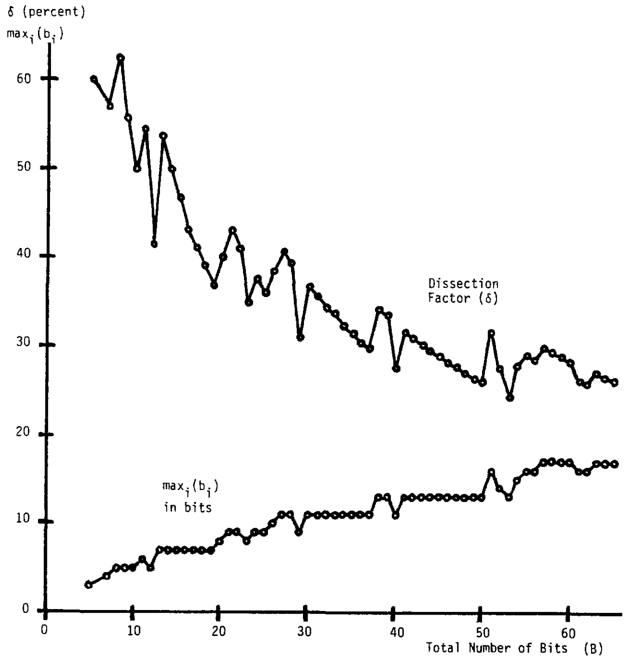


Figure 1—The values of $\max_i(b_i)$ and δ as functions of B

volving the set of moduli p are required (numbers following each operation show the equations where it is used):

1. Subtraction from p_i : (6)
2. Addition modulo p_i : (8)
3. Determination of residues with respect to p_i : (3), (6), (12)
4. Multiplication modulo p_i : (9)
5. Multiplication by p_i : (12)
6. Division by p_i : (10)

We will show that low-cost algorithms exist for performing all of the above operations in a low-cost residue number system. Here, the term "low-cost" refers to the computer implementation of algorithms, keeping in mind that in digital computers addition and subtraction are the fastest and least expensive of the four basic operations, while division is the slowest and most expensive to implement. Most of the operations to be described are also used in encoding, decoding and arithmetic operations for low-cost arithmetic error codes.[6,7]

Subtraction of a b_i-bit binary number x from p_i is quite simple since $p_i=2^{b_i}-1$ is represented in binary as b_i digits of 1. Hence, the digits of p_i-x are the logical complements of the digits of x.

Addition of two b_i-bit binary numbers modulo p_i is also simple. It consists of a simple b_i-bit binary addition with end-around carry; i.e., the carry generated by the last digit position is inserted into the first digit position. This is true since for a sum which is greater than p_i, we have to subtract $p_i=2^{b_i}-1$ in order to obtain its modulo p_i residue. This is done by subtracting 2^{b_i} (discarding the outgoing carry) and adding 1 (inserting a carry into the first digit position). The only problem arises when the sum is equal to p_i, in which case we either need special circuitry to detect this condition and insert a carry into the first digit position if it arises, or simply leave the result as it is and have two representations for zero. This latter approach will cause no difficulty since in all modulo-p_i operations the two values 0 and $2^{b_i}-1$ are entirely equivalent.

To determine the residue of a binary number x with respect to p_i, we simply break x into b_i-bit bytes, starting at the right end, and add the resulting bytes modulo p_i. This is true since the residue of 2^{b_i} with respect to p_i is equal to 1 and the value of x is a polynomial in 2^{b_i}, with the values of the b_i-bit bytes of x as the coefficients. Hence the residue of x with respect to p_i is the same as the residue of the sum of these coefficients with respect to p_i, which is the modulo p_i addition of these coefficients.

Multiplication in digital computers is usually performed through multiple additions, either sequentially by a single adder or in parallel by using a number of carry-save adders.[8] Hence, modulo-p_i multiplication of two numbers can be performed through a number of modulo-p_i additions, the algorithm for which was discussed previously.

Multiplication of a binary number x by $p_i=2^{b_i}-1$ can be done by a single subtraction $x.2^{b_i}-x$, since $x.2^{b_i}$ can be easily obtained through shifting x to the left by b_i bits (inserting b_i zero to the right of x).

Finally, division by p_i (of a number which is a multiple of p_i) can be done by a very interesting algorithm[7] which is obtained by observing that $x=x.2^{b_i}-x.p_i$. Now, the first b_i bits of $x.2^{b_i}$ are known to be zero and since we have $x.p_i$, the first b_i bits of x can be obtained by subtraction. These b_i bits of x now form the second b_i bits of $x.2^{b_i}$ and, hence, the second b_i bits of x are obtained by another subtraction, taking into account a borrow which may have been generated by the first subtraction. This process is continued until all the digits of x are computed.

CONCLUSION

In this paper, we have introduced the class of low-cost residue number representation systems and studied their properties. It appears that such systems alleviate the storage inefficiency normally associated with residue number systems and simplify many of the basic algorithms. The division process, however, remains complex. Therefore, such systems are useful only for special applications.

One disadvantage of the low-cost residue number system is that the moduli, and hence the residues, are larger than those for conventional residue systems with no restriction on p_i's. Therefore, carry propagation delay is not reduced by as much. However, this

TABLE III—Maximum Range of Low-Cost Residue Number Systems with a Given B and with Minimum δ

B	max N	log (max N)	B	max N	log (max N)
5	21	1.322	36	61772533935	10.791
7	105	2.021	37	135899574657	11.133
8	217	2.336	38	265605682657	11.424
9	465	2.667	39	543797467521	11.735
10	651	2.814	40	1050133076895	12.021
11	1953	3.291	41	2184820937985	12.339
12	3255	3.513	42	4202071339935	12.623
13	8001	3.903	43	8764987527681	12.943
14	13335	4.125	44	16832955054495	13.226
15	27559	4.440	45	33731921697439	13.528
16	59055	4.771	46	67729449077535	13.831
17	82677	4.917	47	117830685381465	14.071
18	248031	5.395	48	277472259124095	14.443
19	413385	5.616	49	505978825461585	14.704
20	1003935	6.002	50	1113153416015487	15.047
21	2011807	6.304	51	2233832636833665	15.349
22	4039455	6.606	52	4351417898969631	15.639
23	7027545	6.847	53	8601640032846945	15.935
24	16548735	7.219	54	17792433492601215	16.250
25	30177105	7.480	55	35442210736330753	16.556
26	66389631	7.822	56	71028895618181759	16.853
27	132844159	8.123	57	142904633121912833	17.158
28	266734335	8.426	58	285803715209210623	17.457
29	513010785	8.710	59	575488792479997953	17.761
30	1070075391	9.029	60	1148272730287255039	18.060
31	2055054945	9.313	61	2210621488441664865	18.345
32	4118168929	9.615	62	4572655407598512255	18.660
33	8268764385	9.917	63	9145099114469304447	18.961
34	14385384615	10.158	64	18397371556871432703	19.265
35	33875260545	10.530	65	36368285000677545089	19.561

disadvantage is more than offset by the many advantages which we have enumerated in this paper.

ACKNOWLEDGMENT

The author gratefully acknowledges fruitful discussions with Professor Siavash Shahshahani and also the programming help of Mr. Bijan Aaraam for the preparation of this paper.

REFERENCES

1. Metze, G. and J. E. Robertson, "Elimination of Carry Propagation in Digital Computers," *Proc. of the International Conf. on Information Processing*, pp. 389-396, Paris, June 1959.
2. Avižienis, A., "Signed-Digit Number Representations for Fast Parallel Arithmetic," *IRE Transactions on Electronic Computers*, Vol. EC-10, No. 3, pp. 389-400, September 1961.
3. Svobada, A., "Rational Numerical System of Residue Classes," *Storje Na Zpracovani Informaci*, pp. 9-37, Sbornik V, Nakl. CSAV, Praha, 1957.
4. Svoboda, A., "The Numerical System of Residual Classes in Mathematical Machines," *Proceedings of International Conference on Information Processing*, pp. 419-422, UNESCO, Paris, June 1959, Butterworths, London, 1960.
5. Garner, H. L., "The Residue Number System," *IRE Transactions on Electronic Computers*, Vol. EC-8, No. 8, pp. 140-147, June 1959.
6. Avižienis, A., "Arithmetic Error Codes: Cost and Effectiveness Studies for Application in Digital System Design,"

IEEE Transactions on Computers, Vol. C-20, No. 11, pp. 1322-1331, November 1971.

7. Avižienis, A., "Arithmetic Algorithms for Error-Coded Operands," *IEEE Transactions on Computers*, Vol. C-22, No. 6, pp. 567-572, June 1973.
8. Wallace, C. S., "A Suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers*, Vol. EC-13, No. 1, pp. 14-17, February 1964.

APPENDIX

THEOREMS AND THEIR PROOFS

Theorem 1: $p_i=2^{b_i}-1$ and $p_j=2^{b_j}-1$ are relatively prime if and only if b_i and b_j are relatively prime.

Proof: (Only if part)—Let $b_i=zx$ and $b_j=zy$ with $z>1$. Then, since a^n-1 is divisible by $a-1$, p_i and p_j are both divisible by 2^z-1 and, hence, they are not relatively prime.

(If part)—Suppose there exist pairs of integers of the form $2^{b_i}-1$ and $2^{b_j}-1$ which are not relatively prime while b_i and b_j are. Let 2^x-1 and 2^y-1 be one such pair with $x>y$ and $x+y$ a minimum among all such pairs. Let the odd prime number z divide 2^x-1 and 2^y-1. Then, z must also divide their difference

$$2^x-2^y=2^y(2^{x-y}-1). \qquad (17)$$

Since z cannot divide 2^y, it must divide $2^{x-y}-1$. But now z divides $2^{x-y}-1$ and 2^y-1 with $x-y$ and y relatively prime (since, by assumption, x and y are relatively prime) and $(x-y)+y$ smaller than $x+y$ which was assumed to be a minimum among all such pairs; clearly a contradiction.

Theorem 2: If $b_i\geq 2$ for all i and if for $i\neq j$, we have $b_i\neq b_j$, then

$$\eta=\prod_{i=1}^{k}(1-2^{-b_i})>1-2^{-(\min_{i\leq k}(b_i)-1)}\geq 1/2 \qquad (18)$$

Proof: The second inequality is obvious upon noting that $\min_i(b_i)\geq 2$. To prove the first inequality, we first show, by induction on k, that:

$$\prod_{i=1}^{k}(1-2^{-b_i})\geq 1-\sum_{i=1}^{k}2^{-b_i} \qquad (19)$$

Clearly this is true for $k=1$. To show that if the inequality holds for k it will also hold for $k+1$, we multiply both sides of (19) by the positive value $(1-2^{-b_{k+1}})$ to get:

$$\prod_{i=1}^{k+1}(1-2^{-b_i})\geq 1-\sum_{i=1}^{k+1}2^{-b_i}+b_{k+1}\cdot\sum_{i=1}^{k}2^{-b_i} \qquad (20)$$

The right-hand-side of (20) is clearly greater than the right-hand-side of (19) with k replaced by $k+1$. Next, denoting $\min_{j\leq K}(b_i)$ by m, we write:

$$1-\sum_{i=1}^{k}2^{-b_i}>1-\sum_{x=m}^{\infty}2^{-x}=1-2^{-(m-1)} \qquad (21)$$

Combining (21) with (19), we get the desired result.

Theorem 3: Given $b_1<b_2<\ .\ .\ .\ <b_k$ and $b'_1<b'_2<\ .\ .\ .\ <b'_k$ with $b_1>b'_1$

and

$$\sum_{i=1}^{k}b_i=\sum_{i=1}^{k}b'_i=B, \qquad (22)$$

we have

$$\prod_{i=1}^{k}(2^{b_i}-1)>\prod_{i=1}^{k}(2^{b'_i}-1). \qquad (23)$$

Proof: Using Theorem 2 and the fact that $b'_1\leq b_1-1$, we can write:

$$\begin{aligned}\prod_{i=1}^{k}(2^{b_i}-1)&=2^B\prod_{i=1}^{k}(1-2^{-b_i}) \qquad (24)\\ &>2^B(1-2^{-(b_1-1)})\\ &\geq 2^B(1-2^{-b'_1})\end{aligned}$$

On the other hand:

$$\prod_{i=1}^{k}(2^{b'_i}-1)=2^B\prod_{i=1}^{k}(1-2^{-b'_i})\leq 2^B(1-2^{-b'_1}) \qquad (25)$$

Combining (24) and (25), we get the desired result.

Very fast computation of polynomial remainder sequence coefficient signs*

by JAMES R. PINKERT
University of Tennessee
Knoxville, Tennessee

ABSTRACT

Polynomial remainder sequences are the basis of many important algorithms in symbolic and algebraic manipulation. In a number of these algorithms, the actual coefficients of the sequence are not required; rather, the method uses the signs of the coefficients. Present techniques, however, compute the exact coefficients (or a mixed radix representation of them), and then obtain the signs. This paper discusses a new approach in which interval arithmetic is used to obtain the signs of the coefficients without computing their exact values. Comparisons of this method with analogous standard techniques show empirical computing time reductions of two orders of magnitude for even relatively small cases.

INTRODUCTION

Two occurrences at Mathematical Software II[1] were especially important to this paper. One was Oliver Aberth's presentation of some very interesting techniques in interval arithmetic. The other was a discussion with George E. Collins regarding potential time reductions in algorithms of symbolic and algebraic manipulation if approximate or interval arithmetic could replace some of the exact arithmetic.

One of the author's main interests is the computation of polynomial roots using algebraic algorithms. A major tool in such work is the integer polynomial remainder sequence.[2]

It is important to note that in this application the exact numerical values of coefficients in a given sequence are not required. Rather, one is concerned with the signs of the coefficients.

Two algorithms for computing these signs have received considerable attention.[2] In one, pseudo-division[3] is used to compute a reduced integer polynomial remainder sequence; the signs are then obtained from the integer coefficients. (A greatest common divisor algorithm can be added to obtain a primitive integer polynomial remainder sequence.) In the other algorithm, modular methods are used to obtain mixed radix representations[4] of the coefficients. The signs can then be obtained from these representations without actually computing the integer values.

Since signs, and not numeric values, are required, the considerations brought up in the first paragraph of this section seemed particularly relevant to these computations.

A SAC-1[2] interval arithmetic package was implemented, employing some concepts from Aberth's presentation. The standard SAC-1 algorithm for computing the pseudo-remainder of two polynomials[3] was then modified to use this interval arithmetic.

Empirical comparisons of the standard and modified versions have been very encouraging. Time reductions of two orders of magnitude have been realized on relatively small problems and extrapolations seem to indicate even greater reductions for larger cases. For example, one list of signs which could not be computed in thirty minutes using the standard pseudo-division algorithm was obtained in one second using the interval arithmetic version.

Section two of this paper discusses the interval arithmetic algorithms. The modified polynomial remainder sequence algorithms are presented in Section three. Sample empirical computing times are shown in Section four. Section five summarizes the results.

SAC-1 INTERVAL ARITHMETIC

This section discusses the SAC-1 interval arithmetic package used to compute reduced polynomial remainder sequences. Some initial comments must be made regarding these algorithms.

First, they are included for completeness of discussion and better evaluation of results presented in Section four, not on any premise of significant developments in themselves. These algorithms were designed for efficient implementation in SAC-1, and hence to expedite testing of the hypothesis that interval arith-

* Research supported by the University of Tennessee Faculty Research Fund.

metic could be used to improve polynomial remainder sequence generation. Given the successful test results, effort can now be directed to refinement and expansion of this package to form a viable SAC-1 Interval Arithmetic Module.

Second, and related to the previous paragraph, only the fundamental operations of the algorithms are shown. Actual subroutine calls, list operations, and checks for such things as zero inputs and null lists are not included. A complete listing and/or card deck of the actual system can be obtained from the author.

Finally, some notational conciseness has not been used, in the interest of clarity (e.g., deeply nested if-then-else statements).

Consider an integer $A \neq 0$, where for some integer $\beta > 1$, $\beta^m \leq |A| < \beta^{m+1}$. Then A can be written as $A = \sum_{j=0}^{m} \alpha_j \beta^j$ where $\alpha_j = 0$ or $\text{sign}\{\alpha_j\} = \text{sign}\{A\}$, $|\alpha_j| < \beta$, and $\alpha_m \neq 0$. The SAC-1 Integer Arithmetic System[5] uses this technique, representing A as the list $(a_0, a_1, \ldots, a_m)$ where a_j is the Fortran representation of α_j.

Now consider the interval $I = [\beta^\chi\{A-\epsilon\}, \beta^\chi\{A+\epsilon\}]$ where A is as above, $0 \leq \epsilon < \beta$, and $0 \leq \chi < \beta$. This interval is represented in SAC-1 as the list $((a_0, a_1, \ldots, a_m), e, x)$ where $\bar{A} = (a_0, a_1 \ldots, a_m)$ is the representation of A, e is the Fortran representation of ϵ, and x is the Fortran representation of χ.

The interval I is spoken of as "carried to m+1 β-digits." For conciseness, it will be written $\beta^\chi[A \pm \epsilon]$.

The Integer Arithmetic System representation of $A = 0$ is the null list, (). The representation of interval $I = [0, 0]$ will also be the null list.

The following algorithm is used for interval subtraction, $I - J$, where $I = \beta^{\chi_1}[A \pm \epsilon_1] = ((a_0, a_1, \ldots, a_{d_1}), e_1, x_1) = (\bar{A}, e_1, x_1)$, $J = \beta^{\chi_2}[B \pm \epsilon_2] = ((b_0, b_1, \ldots, b_{d_2}), e_2, x_2) = (\bar{B}, e_2, x_2)$, and the result is carried to n β-digits.

Subtraction:

```
Step 1: If $x_1 = x_2$ then $x_3 \leftarrow x_1$;
Step 2: If $x_1 > x_2$ then do;
          $e_2 \leftarrow 2$;
          $\bar{B} \leftarrow (b_{x_1-x_2}, b_{x_1-x_2+1}, \ldots, b_{d_2})$;
          $x_3 \leftarrow x_1$;
          end;
Step 3: If $x_2 > x_1$ then do;
          $e_1 \leftarrow 2$;
          $\bar{A} \leftarrow (a_{x_2-x_1}, a_{x_2-x_1+1}, \ldots, a_{d_1})$;
          $x_3 \leftarrow x_2$;
          end;
Step 4: $e_3 \leftarrow e_1 + e_2$;
Step 5: $\bar{C} \leftarrow \bar{A} - \bar{B} = (c_0, c_1, \ldots, c_{d_3})$;
Step 6: If $e_3 \geq \beta$ then do;
          $\bar{C} \leftarrow (c_1, c_2, \ldots, c_{d_3})$;
          $h \leftarrow 1$;
          $e_3 \leftarrow 2$;
          $x_3 \leftarrow x_3 + 1$;
          end;
        else $h \leftarrow 0$;
Step 7: If $d_3 - h + 1 > n$ then do;
          $m \leftarrow d_3 - n + 1$;
          $\bar{C} \leftarrow (c_m, c_{m+1}, \ldots, c_{d_3})$;
          $e_3 \leftarrow 2$;
          $x_3 \leftarrow x_3 + m - h$;
          end;
Step 8: Return $C = (\bar{C}, e_3, x_3)$.
```

Step 1 through Step 3 modify exponents if necessary. For example, if $x_2 > x_1$ then $\bar{A}$ would be modified to $((a_{x_2-x_1}, a_{x_2-x_1+1}, \ldots, a_{d_1}), 2, x_2)$. Since $p = \sum_{j=0}^{\chi_2-\chi_1-1} |\alpha_j \beta^j| < \beta^{\chi_2-\chi_1}$ and $\epsilon_2 < \beta$, it follows that $p + \epsilon_2 < 2\beta^{\chi_2-\chi_1}$ and $\beta^{\chi_1}\left[\sum_{j=0}^{d_1}\{\alpha_j\beta^j\} \pm \epsilon_2\right] \subseteq \beta^{\chi_2}\left[\sum_{j=x_2-x_1}^{d_1}\{\alpha_j\beta^j\} \pm 2\right]$. Note that one could have $x_2 - x_1 > d_1$, in which case the result is $((), 2, x_2) \equiv \beta^{\chi_2}[\pm 2]$.

Step 6 insures that the resulting error term will satisfy the definition, i.e., $e_3 < \beta$. Step 7 insures that the interval will be carried to no more than the specified number, n, of β-digits.

In multiplying two intervals, the system does three multiplies of a β-digit error term, say c, times a β-integer, say $\bar{B}$. Each result is a single β-digit error term, say e, times a power of β, i.e., $|c\bar{B}| \leq e\beta^\chi$. The following algorithm performs this operation. Inputs are β-digit c and β-integer $\bar{B} = (b_0, b_1, \ldots, b_{d_2})$.

Special Multiplication, mpys:

```
Step 1: $G \leftarrow 1 + |b_{d_2-2}| + c|b_{d_2-1}| + c|b_{d_2}|\beta =$
          $(g_0, g_1, \ldots, g_k)$;
Step 2: If $k = 2$ then do;
          $x \leftarrow d_2 + 1$;
          $e \leftarrow g_2 + 1$;
          end;
        else do;
          $x \leftarrow d_2$
          $e \leftarrow g_1 + 1$;
          end;
Step 3: If $e = \beta$ then do;
          $x \leftarrow x + 1$;
          $e \leftarrow 1$;
          end;
Step 4: Return $C = (e, x)$.
```

Note that $c|\bar{B}| = c\sum_{j=0}^{d_2} |b_j|\beta^j = c\sum_{j=0}^{d_2-2} |b_j|\beta^j + c|b_{d_2-1}|\beta^{d_2-1} + c|b_{d_2}|\beta^{d_2} < c(|b_{d_2-2}|+1)\beta^{d_2-2} + c|b_{d_2-1}|\beta^{d_2-1} + c|b_{d_2}|\beta^{d_2} < (|b_{d_2-2}| + 1)\beta^{d_2-1} + c|b_{d_2-1}|\beta^{d_2-1} + c|b_{d_2}|\beta^{d_2} \leq (\beta-1)\beta^{d_2+1} + \beta^{d_2-1}$. Hence Step 1 of the algorithm will give a result which bounds $c|\bar{B}|$, and which has $k=1$ or $k=2$.

Step 2 sets the exponent and rounds up the error term. Step 3 insures that the error term is a single β-digit.

The following algorithm uses mpys to perform interval multiplication.

Multiplication:

```
Step 1:  $E_1 \leftarrow \text{mpys}\{a\bar{B}\} = (e_1, f_1)$ ;
Step 2:  $E_2 \leftarrow \text{mpys}\{b\bar{A}\} = (e_2, f_2)$ ;
Step 3:  $E_3 \leftarrow \text{mpys}\{a(b)\} = (e_3, f_3)$ ;
Step 4:  $G \leftarrow e_1\beta^{f_1} + e_2\beta^{f_2} + e_3\beta^{f_3} = (g_0, g_1, \ldots, g_f)$ ;
Step 5:  $e_4 \leftarrow g_f + 1$ ;
Step 6:  If $e_4 = \beta$ then do;
           $e_4 \leftarrow 1$ ;
           $f \leftarrow f + 1$ ;
           end;
Step 7:  $\bar{C} \leftarrow \bar{A} \bullet \bar{B} = (c_0, c_1, \ldots, c_{d_4})$ ;
Step 8:  If $f > 0$ then do;
           $e_4 \leftarrow e_4 + 1$ ;
           If $e_4 = \beta$ then do
                $e_4 \leftarrow 2$ ;
                $f \leftarrow f + 1$ ;
                end;
           end;
Step 9:  If $d_4 - f + 1 > n$ then do:
           $e_4 \leftarrow 2$ ;
           $f \leftarrow d_4 - n + 1$ ;
           end;
Step 10: $x_4 \leftarrow f + x_1 + x_2$ ;
Step 11: $\bar{C} \leftarrow (c_f, c_{f+1}, \ldots, c_{d_4})$ ;
Step 12: Return $(\bar{C}, e_4, x_4)$.
```

Step 1 through Step 5 do the standard error term multiplies, add the results, and round up the final error term. Step 6 insures that the final error term is a single β-digit.

Step 8 checks whether the error has a positive exponent. If it does, some digits of the product $\bar{C}$ must be dropped and the error term correspondingly increased by one. The increased error term must then be checked to insure that it is a single β-digit.

Step 9 computes the length of the product, to see that it is carried to no more than the specified number of β-digits.

PSEUDO-REMAINDERS AND SIGN LISTS

This section describes an algorithm for computing the pseudo-remainder of two polynomials, and an algorithm which uses pseudo-remainders to compute one standard type of coefficient sign list. The discussion is abbreviated, since the methods have appeared previously.[2,3]

A SAC-1 interval polynomial has exactly the same internal representation as an integer polynomial,[3] except that the numerical coefficients are intervals rather than integers. Hence the pseudo-remainder algorithm can be applied to either type of polynomial simply by calling the corresponding arithmetic routines.

The following algorithm computes the pseudo-remainder of two polynomials, P(v) and Q(v). Note that "deg," "ldcf," and "red" return the degree, the leading coefficient, and the reductum, respectively.

Pseudo-remainder, psrem:

```
Step 1:  $k \leftarrow \deg\{P\} - \deg\{Q\} + 1$ ;
Step 2:  For $j \leftarrow 1$ to k by 1 do;
           $d \leftarrow \deg\{P\} - \deg\{Q\}$ ;
           if $d < 0$ then $R \leftarrow \text{ldcf}\{Q\} \cdot P$
                else $R \leftarrow \text{ldcf}\{Q\} \cdot \text{red}\{P\} -$
                     $\text{ldcf}\{P\} \cdot \text{red}\{Q\} \cdot v^d$ ;
           $P \leftarrow R$ ;
           end;
Step 3:  Return R.
```

The next algorithm applies pseudo-remainders to compute the list of degrees and leading coefficient signs in the reduced polynomial remainder sequence for polynomials P and Q.

List of Signs and Degrees, lsad:

```
Step 1:  $L \leftarrow (\text{sign}\{\text{ldcf}\{P\}\}, \deg\{P\}, \text{sign}\{\text{ldcf}\{Q\}\},$
         $\deg\{Q\})$ ;
Step 2:  While $(R \neq 0)$ & $(d \neq 0)$ & $(0 \notin r)$ do;
         $R \leftarrow \text{psrem}\{P, Q\}$ ;
         if $R \neq 0$ then do;
              $r \leftarrow \text{ldcf}\{R\}$ ;
              if $0 \notin r$ then do;
                   $s \leftarrow \text{sign}\{r\}$ ;
                   $d \leftarrow \deg\{R\}$ ;
                   $L \leftarrow L \| (s, d)$ ;
                   end;
                else $L \leftarrow ()$ ;
              end;
         $P \leftarrow Q$ ;
         $Q \leftarrow R$ ;
         end;
Step 3:  Return L.
```

Note that there are two return conditions. A successful return is executed when the pseudo-remainder is zero or is of degree zero. A failure return is executed when the sign of the leading coefficient cannot be determined. By definition, if $R \neq 0$ then $r = \text{ldcf}\{R\} \neq 0$. Hence if $0 \in r$ then the failure condition has occurred. The cause, of course, is that the intervals have not been carried to a sufficient number of β-digits.

The latter algorithm can be used with integer arithmetic simply by removing the test for $0 \in r$:

```
Step 2:  While $(R \neq 0)$ & $(d \neq 0)$ do;
         $R \leftarrow \text{psrem}\{P, Q\}$ ;
         if $R \neq 0$ then $L \leftarrow L \| (\text{sign}\{\text{ldcf}\{R\}\}, \deg$
           $\{R\})$ ;
         $P \leftarrow Q$ ;
         $Q \leftarrow R$ ;
         end;
```

COMPARISON OF EMPIRICAL COMPUTING TIMES

The algorithm for computing the list of signs and degrees, lsad, was applied to randomly generated poly-

TABLE I

$L(|P|_\infty), L(|Q|_\infty)=5$

		degree of Q				
degree of P	P′	5	4	3	2	1
6	82086	81674	26150	7062	2038	1020
	822	931	607	424	332	203
5	16407		14743	4559	1506	707
	590		416	416	241	138
4	3195			3122	1040	441
	324			291	183	91
3	636				519	274
	133				103	83
2	141					150
	50					33

TABLE III

$L(|P|_\infty), L(|Q|_\infty)=15$

		degree of Q				
degree of P	P′	5	4	3	2	1
6	?	?	?	?	?	?
	965	858	690	607	316	250
5	131714		142139	38763	11140	4768
	516		566	433	292	150
4	22764			22423	6614	2515
	350			317	191	100
3	3969				3944	1423
	125				133	83
2	757					683
	50					42

nomials, first using integer methods and then using interval methods. The empirical results are shown in this section.

Random polynomial P, of specified degree and infinity norm, was obtained. Since lsad{P, P′} is used often,[2] this case was run first. Then, for $1 \leq j < \deg\{P\}$, random polynomial Q of degree j was generated, and lsad{P, Q} was computed.

This process was repeated for another polynomial P of the same degree and norm. The exact number of repetitions for a given entry depended on the time required for each repetition.

The average results are shown in Tables I, II, III, and IV. In each comparison, the first entry of the pair is the time (in milliseconds) required for the integer method. The second entry is the time (in milliseconds) required for the interval method, using the minimum possible number of β-digits. A question mark indicates that the set of runs could not be completed in a reasonable amount of time.

TABLE II

$L(|P|_\infty), L(|Q|_\infty)=10$

		degree of Q				
degree of P	P′	5	4	3	2	1
6	353547	335427	101117	27564	7572	3470
	949	866	783	707	325	241
5	61244		56543	17614	4884	2147
	541		475	292	217	133
4	10766			10725	3437	1423
	400			250	242	117
3	1939				1889	774
	100				125	67
2	341					349
	50					50

TABLE IV

$L(|P|_\infty), L(|Q|_\infty)=20$

		degree of Q				
degree of P	P′	5	4	3	2	1
6	?	?	?	?	?	?
	940	732	649	541	366	225
5	?		?	?	?	?
	541		566	391	250	133
4	41958			42216	11781	4568
	275			300	208	108
3	6898				6623	2438
	125				133	84
2	1157					1206
	59					33

The time comparisons were made with reduced rather than primitive integer techniques because of the direct analogy in algorithms. Modular methods were not included because, unlike many other instances, they have not resulted in any consistent improvements over integer methods.[2]

Note that the times shown are with intervals carried to the minimum number of β-digits. How does one determine this minimum? As yet, the author has found no way of deriving any theoretical answers to the question. For the sample runs discussed, the minimums were determined experimentally. Given this situation, the times shown for the interval method should include overhead resulting from using too few or too many β-digits.

A small percentage of the test runs required five β-digits; all others required four or fewer. This result is very interesting, since coefficients of a hundred β-digits were not uncommon in the integer computations. It suggests an algorithm which simply starts with one β-digit, and increments by one until the list is computed. This approach was tried, and it gave an average increase of two to three times the illustrated results.

The observed minimum number of β-digits seemed to depend mainly on the degree of Q (or the degree of P'). Hence the author started at this value and doubled the number of β-digits used each time a failure occurred. This approach yielded almost exactly the same set of times as shown in the tables.

SUMMARY

Interval methods appear to have great potential in the computation of polynomial remainder sequence coefficient signs.

Obviously, more reseach needs to be done in determining how many β-digits to carry for the intervals. However, significant improvements seem to be possible even with simplistic approaches.

It is hoped that these results, in addition to being useful per se, might suggest further applications of interval methods in symbolic and algebraic manipulation.

REFERENCES

1. *Proceedings of Mathematical Software II*, Purdue University, May 1974.
2. Pinkert, J. R., *Algebraic Algorithms for Computing the Complex Zeros of Gaussian Polynomials*, University of Wisconsin Computer Sciences Department, Ph.D. Thesis, June 1973.
3. Collins, G. E., *The SAC-1 Polynomial System*, University of Wisconsin Computer Sciences Department Technical Report No. 115, March 1971.
4. Knuth, D. E., *The Art of Computer Programming, Vol. II: Seminumerical Algorithms*, Addison-Wesley Publishing Company, Reading, Mass., 1969.
5. Collins, G. E., and J. R. Pinkert, *The Revised SAC-1 Integer Arithmetic System*, University of Wisconsin Computing Center Technical Report No. 9, November 1968.

System theoretic implications of numerical methods applied to the solution of ordinary differential equations

by T. G. WINDEKNECHT and H. D'ANGELO
Memphis State University
Memphis, Tennessee

ABSTRACT

This paper presents a system-theoretic analysis of numerical methods used in approximating solutions of ordinary differential equations. By representing ordinary differential equations with system block diagrams (i.e., interconnections of static elements and integrators), a numerical method can be viewed as a process in which the integrators of a continuous system are replaced by discrete approximations to the integrators (i.e., by discrete subsystems made up of interconnections of static elements and delays). The main result of the paper establishes that if the system block diagram corresponding to the original differential equation has no static loops and if the discrete subsystems used to replace the integrators have no static loops and no static through paths, then the resulting discrete system can be characterized by explicit difference equations. A system-theoretic study is conducted of several of the more commonly used numerical methods (the Euler, trapezoidal, Runge-Kutta, and predictor-corrector methods) and the limitations of using these numerical methods in the real-time analysis of input-output systems are examined.

INTRODUCTION

Important insight into numerical methods used in the solution of ordinary differential equations can be obtained by studying these numerical methods from a system theoretic standpoint. The system block diagram (an interconnection of static and dynamic elemental components) has proved to be particularly helpful in this regard. With this point of view, a useful categorization of numerical methods is obtained.

THE PROBLEM

The problem addressed in this paper is the following: Determine a numerical solution to the system of ordinary differential equations given in the standard canonical form

$$\frac{dx(t)}{dt} = f(x(t),u(t)),\ x(t_o) = x_o,\ t\epsilon[t_o,t_f] \qquad (1)$$

where x is an n-vector, u is an m-vector, and f is a function such that $f: R^{n+m} \rightarrow R^n$.

The ultimate objective in applying a numerical method is to compute a sequence

$$\hat{x}(t_o), \hat{x}(t_o+\delta), \hat{x}(t_o+2\delta), \ldots, \hat{x}(t), \ldots, \hat{x}(t_f) \qquad (2)$$

which is, in some sense, a good approximation to the sequence

$$x(t_o), x(t_o+\delta), x(t_o+2\delta), \ldots, x(t), \ldots, x(t_f) \qquad (3)$$

where the continuous variable x, defined on the interval $[t_o,t_f]$ is the solution of equation (1).

It is generally computationally advantageous to be able to compute sequence (2) as a solution to a set of difference equations in the standard state-variable form

$$v(t+\delta) = f_a(v(t),u(t)) \qquad (4a)$$

$$,\ v(t_o) = g(x_o),\ t\epsilon Td$$

$$x(t) = E\, v(t) \qquad (4b)$$

where x is an n-vector, v is an N-vector $(N \geq n)$, u is an m-vector, and f_a is a function such that $f_a: R^{N+m} \rightarrow R^N$, g is a function such that $g: R^n \rightarrow R^N$, E is an nxN matrix of rank n whose elements are either 0 or 1 with one and only one 1 in each row (thus, the components of x are a subset of the components of v), and T_d is the discrete time set $\{t_o, t_o+\delta, t_o+2\delta, \ldots, t_f\}$.

The form of difference equations (4), in which f_a does not depend on $v(t+\delta)$, is said to be *explicit.* Explicit difference equations are computationally attractive since, for most functions f_a, they allow the approximation sequence (2) to be generated in a straightforward iterative fashion. More precisely:[1]

Theorem 1 Explicit difference equation (4) has a unique solution provided only that f_a is defined at each stage of iteration. The solution can be generated on a computer provided that f_a is a computer function and that $|v(t+\delta)|$ is no larger than the largest number in the computer.

However, under certain severe modeling constraints, it may be necessary to settle for less attractive numerical methods in which the approximation sequence (2) is computed as a solution to a set of difference equations of the form

$$v(t+\delta)=f_s(v(t),v(t+\delta),u(t)) \quad , v(t_0)=g(x_0),\ t\epsilon T_d \tag{5a}$$

$$x(t)=E\,v(t) \tag{5b}$$

where f_s is a function such that $f_s:R^{2N+m}\rightarrow R^N$. Difference equations of this form, in which f_s depends on $v(t+\delta)$, are said to be *implicit.* The computational difficulties associated with implicit difference equations stem from the fact that, depending on the nature of the function f_s, it may not be possible to solve for $v(t+\delta)$ and thus obtain the desirable recursive form of equations (4). In such cases, where the solutions are known to exist, and they may not exist, computing $v(t+\delta)$ from a knowledge of $v(t)$ and $u(t)$ is not straightforward. Often, an iterative method of solution, such as a Newton method, with generally unknown convergence properties, is required to approximate $v(t+\delta)$. Certainly, one should avoid this latter form if possible.

SYSTEM BLOCK DIAGRAMS

In system theory it is sometimes convenient to represent sets of equations characterizing dynamic systems by interconnections of elemental components. Systems characterized by differential equations, and thus defined on the continuous time set $[t_0,t_f]$, $t_f>t_0$, are called *continuous systems*; systems characterized by difference equations, and thus defined on the discrete time set T_d, are called *discrete systems.* A sufficient set of elemental components for synthesizing the necessary interconnections consists of the general static element and two fundamental dynamic elements, the delay and the integrator. Each elemental component defines a causal relationship between an *output variable* and one or more *input variables.* The set of elemental components is defined:

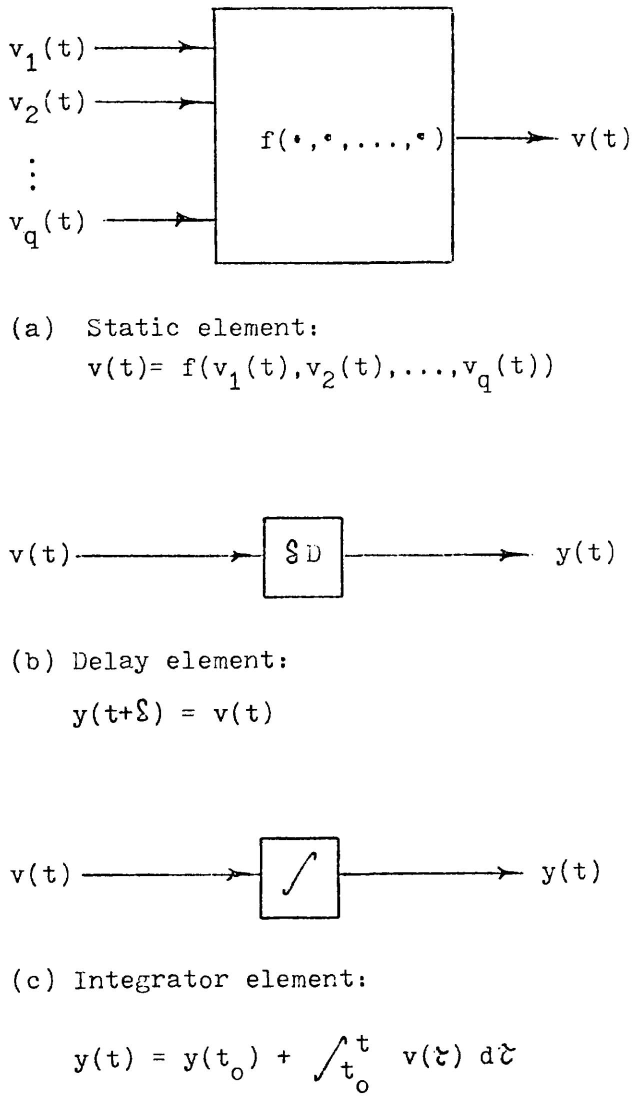
a. A general static element, represented graphically in Figure 1a, is a functional relationship between q input variables and an output variable. It has the form

$$v(t)=f(v_1(t),v_2(t),\ldots,v_q(t)) \tag{6}$$

where f is a function such that $f:R^q\rightarrow R$.

b. A δ-delay element, represented in Figure 1b, is a single-input, single-output element satisfying the quasi-functional relationship between two dynamic variables v and y

$$y(t+\delta)=v(t),\ t\epsilon T_d \tag{7}$$



(a) Static element:
$v(t)= f(v_1(t),v_2(t),\ldots,v_q(t))$

(b) Delay element:
$y(t+\delta) = v(t)$

(c) Integrator element:
$y(t) = y(t_0) + \int_{t_0}^{t} v(\tau)\, d\tau$

Figure 1—Fundamental elemental components

c. An integrator element, represented in Figure 1c, is a single-input, single-output element defined so that the output is the integral of the input (or, equivalently, the input is the derivative of the output):

$$y(t)=y(t_0)+\int_{t_0}^{t} v(\tau)\,d\tau \tag{8}$$

Note that the initial condition $y(t_0)$, as well as the input variable v on the interval (t_0,t), is required to determine the output $y(t)$.

A fixed-structure discrete dynamic system is any *consistent* interconnection of delays and static elements. An interconnection is consistent if no two outputs of elemental components are connected together (i.e., no two outputs should represent the same variable). Similarly, a fixed-structure continuous dynamic system is any consistent interconnection of integrators and static elements. Such interconnections of elemental systems are called *system block diagrams.*

It is important to establish the conditions under which system block diagrams are characterized by canonical state variable equations, i.e., by equations of the form

$$\begin{aligned} x(t+\delta) &= f(x(t),u(t)) \\ y(t) &= g(x(t),u(t)) \end{aligned}, \; t\epsilon T_d \qquad (9)$$

for discrete systems or by equations of the form

$$\begin{aligned} \frac{dx(t)}{dt} &= f(x(t),u(t)) \\ y(t) &= g(x(t),u(t)) \end{aligned}, \; t\epsilon[t_o,t_f] \qquad (10)$$

for continuous systems. Toward this end, the concept of a *proper interconnection* is introduced. An interconnection is proper provided every closed path (in the direction of the arrows) on the system block diagram contains at least one delay element (or, in the case of a continuous system, one integrator). A closed path that does not contain a delay (or integrator) is called a *static loop.* Thus, a system with no static loops is a proper interconnection.

Two important results relating to system block and canonical state equations are found in References 2 and 3.

Theorem 2 Every proper interconnection of static elements and delays admits a state-variable characterization of the form of equations (9); every proper interconnection of static elements and integrators admits a state-variable characterization of the form of equations (10).

Theorem 3 Every set of difference equations in state-variable canonical form (equations (9)) can be represented by a proper interconnection of static elements and delays; every set of differential equations in state-variable canonical form (equations (10)) can be represented by a proper interconnection of static elements and integrators.

COMPUTATIONAL IMPLICATIONS OF STATIC LOOPS

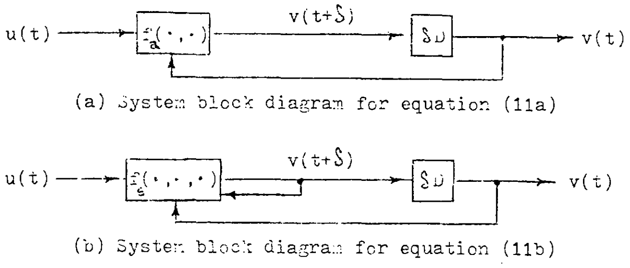
Consider the system block diagrams in Figure 2 corresponding to the following two first-order difference equations, the first explicit and the second implicit:

$$v(t+\delta) = f_a(v(t),u(t)) \qquad (11a)$$

$$v(t+\delta) = f_s(v(t),v(t+\delta), u(t)) \qquad (11b)$$

Note that the explicit first-order difference equation can *always* result in a proper system block diagram (i.e., one without static loops), whereas the implicit first-order difference equation *always* results in a system interconnection with a static loop.

For higher order systems, the results are almost the same: Explicit difference equations (4a) can *always* result in a proper system block diagram, whereas implicit difference equations (5a) *generally* result in system interconnections with a static loop. The exceptional case where an implicit difference equation does not result in any static loops is the somewhat trivial case where a simple reindexing of variables renders the set of difference equations in the following "lower triangular form":



(a) System block diagram for equation (11a)

(b) System block diagram for equation (11b)

Figure 2—System block diagrams for explicit and implicit first-order difference equations

$$\begin{aligned} v_1(t+\delta) &= f_1(v(t),u(t)) \\ v_2(t+\delta) &= f_2(v(t),v_1(t+\delta),u(t)) \\ &\vdots \\ v_i(t+\delta) &= f_i(v(t),v_1(t+\delta),v_2(t+\delta),\ldots, \\ &\qquad v_{i-1}(t+\delta),u(t)) \\ &\vdots \\ v_N(t+\delta) &= f_N(v(t),v_1(t+\delta),v_2(t+\delta),\ldots, \\ &\qquad v_{N-1}(t+\delta),u(t)) \end{aligned} \qquad (12)$$

where v is an N-vector consisting of the scalar components $v_1,v_2,\ldots,v_N$, and f_i, $i=1,2,\ldots,N$, is a function such that $f_i:R^{N+i}\rightarrow R$. Note that this is a rather exceptional implicit form in that

a. The sequence $v(t_o),v(t_o+\delta),\ldots,v(t_f)$ can be computed in a simple iterative fashion.

b. This form admits a system block diagram representation with no static loops.

That the above statements a and b are equivalent statements is easily established by showing that equations (12) can always be put into explicit form by substituting the first equation into the second equation to eliminate $v_1(t+\delta)$, then substituting the first and new second equation into the third equation to eliminate $v_1(t+\delta)$ and $v_2(t+\delta)$, etc. Therefore, the computationally desirable form can be associated with system block diagrams having no static loops and, with the exception of the "lower triangular forms", the computationally difficult implicit form can be associated with block diagrams having at least one static loop.

DISCRETE APPROXIMATIONS TO CONTINUOUS SYSTEMS

It can be shown that every ordinary differential equation in canonical state-variable form can be represented by a proper system block diagram (i.e., an interconnection of static elements and integrators containing no static loops). For example, the first-order differential equation

$$\frac{dx(t)}{dt}=f(x(t),u(t)) \qquad (13)$$

is represented by the system block diagram shown in Figure 3. Similarly, the set of two simultaneous first-order differential equations

$$\frac{dx_1(t)}{dt}=f_1(x_1(t),x_2(t),u(t))$$
$$\frac{dx_2(t)}{dt}=f_2(x_1(t),x_2(t)) \qquad (14)$$

is represented by the system block diagram shown in Figure 4.

One may, taking a system-theoretic viewpoint, interpret most numerical methods for approximating solutions to ordinary differential equations to be equivalent to substituting discrete subsystems for the integrators in the system, thus converting the continuous system to a discrete system. In some numerical methods integrators are replaced by single-input, single-output discrete subsystems on an individual basis whereas in other numerical methods all the integrators are considered to make up a single subsystem and this subsystem is replaced as a single entity. In the latter case, the n integrators in a system are considered to be an n-input, n-output subsystem consisting of n parallel non-interacting integrators; this subsystem is then replaced by an n-input, n-output discrete subsystem. We speak of these subsystems which replace the integrators in a system as discrete approximations to integration or simply as discrete integrators. If the resulting discrete system has no static loops, such a substitution for the integrators in the system leads to explicit difference equations (4); if static loops are present then implicit difference equations (5) are obtained. As far as computational efficiency is concerned, we prefer those discrete approximations to integration which result in discrete systems with no static loops. Since one can always obtain a system block diagram with no static loops to characterize the canonical state-variable differential equations, a condition for the discrete approximations to integration sufficient to assure that the resulting discrete system also has no static loops is easily obtained.

Theorem 4 If the system block diagrams of the discrete approximations to the integrators in a proper system block diagram of a continuous system have

(i) no static loops
(ii) no static through paths (i.e., paths from input to output without a delay element)

then the system block diagram of the resulting discrete system is also proper and can therefore be characterized by the set of explicit difference equations (4).

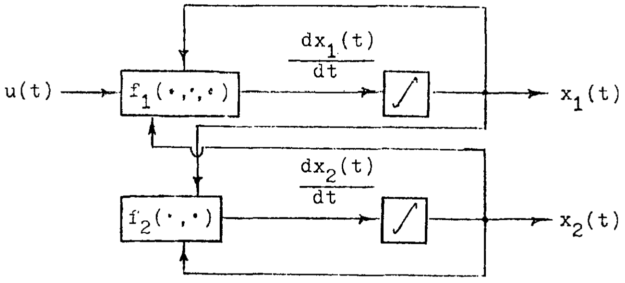


Figure 4—System block diagram for equations (13)

As an example, consider using the Euler zero-order approximation to integration[4] in obtaining a numerical solution to the following first-order differential equation:

$$\frac{dx(t)}{dt}=\cos(x(t)\ u(t)),x(0)=1 \qquad (15)$$

The system block diagram corresponding to equation (15) is shown in Figure 5. An integrator with input v(t) and output x(t), shown in Figure 6a, is characterized by the equation

$$x(t+\delta)=x(t)+\int_t^{t+\delta}v(\tau)d\tau \qquad (16)$$

The Euler method approximates the definite integral by a rectangle:

$$\int_t^{t+\delta}v(\tau)d\tau\cong\delta v(t) \qquad (17)$$

Thus, using the Euler approximation in equation (16) gives the discrete approximation to integration:

$$x(t+\delta)=x(t)+\delta v(t) \qquad (18)$$

The system block diagram for this discrete approximation to integration is shown in Figure (6b). Note that

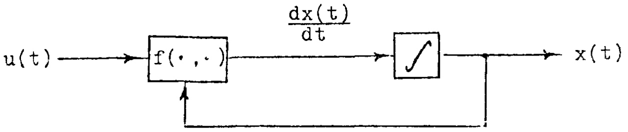


Figure 3—System block diagram for equation (13)

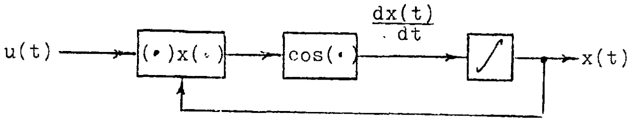


Figure 5—System block diagram for equation (15)

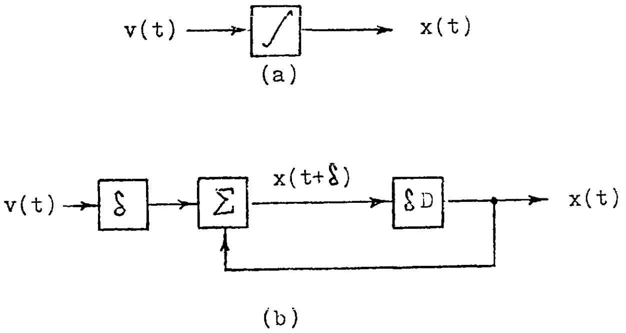


Figure 6—An integrator, (a), and the Euler discrete approximation to it, (b)

the Euler discrete approximation to integration has neither static loops nor static through paths.

Replacing the integrator in the system block diagram of Figure 5 by the Euler discrete approximation to it, Figure 6b, results in the discrete system shown in Figure 7. Importantly, the resulting discrete system has no static loops and is thus characterized by an explicit difference equation:

$$x(t+\delta)=x(t)+\delta\cos(x(t)\ u(t)) \qquad (18)$$

Some other well-known discrete approximations to integration and their system theoretic interpretations are given in the following section.

EXAMPLES OF SOME FREQUENTLY USED NUMERICAL METHODS

In this section some well-known numerical methods will be studied from a system theoretic viewpoint. Much of the literature on the numerical analysis of ordinary differential equations deals with the problem of finding solutions to the class of differential equations defined by

$$\frac{dx(t)}{dt}=f(x(t),t) \qquad (18a)$$

Note that the class of differential equations defined by equation (18a) is somewhat narrower than the class defined by equation (1). Specifically, equation (1) is equivalent to equation (18a) only for the special case that

$$u(t)=t$$

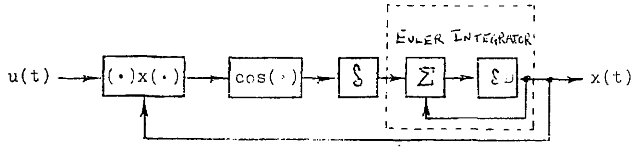


Figure 7—The discrete approximation to equation (15)

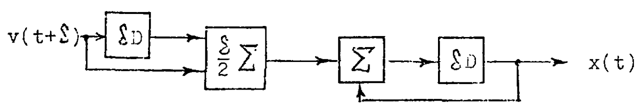


Figure 8—System block diagram for equation (20)

From a systems standpoint, this is unfortunate since much of system theory deals with systems defined in an input-output sense and restricting all inputs u to be defined by u(t)=t is quite unsatisfactory. As we shall see, the problem of extending a numerical method designed for equation (18a) so that it can be used for equation (1) is not always straightforward and, in a systems sense, not always possible. Some of the examples in this section will illustrate this.

For simplicity, but with no loss of generality, the examples will deal with finding solutions of a first-order differential equation, i.e., equation (13).

(i) *The trapezoidal rule:* Here the definite integral is approximated by a trapezoid:

$$\int_t^{t+\delta} v(\tau)\,d\tau=\frac{\delta}{2}(v(t)+v(t+\delta)) \qquad (19)$$

Thus, the relation between the input v and the output x of the trapezoidal approximation to integration is

$$x(t+\delta)=x(t)+\frac{\delta}{2}(v(t)+v(t+\delta)) \qquad (20)$$

A system block diagram of the system characterized by equation (20) is shown in Figure 8. The difficulty with this system is that although it provides x(t) as an output, it requires v(t+δ) as an input; v(t) should be the input if this discrete system is to be used as a substitution for integration. This difficulty can be resolved by introducing a time shift in the original continuous system which is to be discretized for computational purposes. For example, the continuous system shown in Figure 5 can be relabeled with the necessary time shift as shown in Figure 9. With such a time shift we can consider using a discrete approximation to an integrator with input v(t+δ) and output x(t+δ). Such a subsystem is easily achieved with the trapezoidal approximation to integration of Figure 8 simply by considering the output to be the input of the rightmost delay element (i.e., x(t+δ) rather than the output of that delay (i.e., rather than x(t)). Figure 10 shows this rearranging.

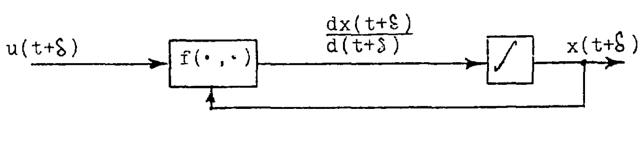


Figure 9—Time-shifted system block diagram for system of Figure 3

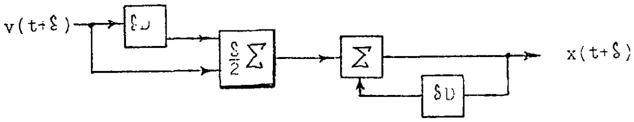


Figure 10—System block diagram for trapezoidal integrator (Figure 8) redrawn to show x(t=δ) as the output

It can be seen that with v(t+δ) as the input and x(t+δ) as the output, the trapezoidal approximation to integration has a static through path. Substituting the trapezoidal integrator into the time-shifted original continuous system results in a discrete system with one static loop (Figure 11) which in turn results in implicit difference equations.

Note that two delay elements are used in the approximation to one integrator. Thus, two initial values are required for the resulting discrete system and the single initial value provided for the first-order continuous system is not sufficient to provide a unique solution for the approximating discrete system. For example, if the discrete process is started at t = −δ, one must have the two initial values f(x(0),u(0)) and x(0). In this case, computing the initial value f(x(0), u(0)) from the original initial value x(0) and the initial value of the input u(0) is relatively straightforward. However, in a systems sense, the time shift from t to t+δ is a severe modification. As a result of the time shift, the discrete system operates in *future time* (i.e., at time t, the system requires input u(t+δ)). If such an approximating system were required to operate in *real time*, then the time shift would represent an impossible realization. Of course one may decide to use the system, which as an interconnection of elements is realizable, in real time nevertheless. This implies shifting time back again from t+δ to t and starting the process at t=0. The difficulty with this is that the initial values required now are x(δ) and f(x(δ),u(δ)) which, of course, are not generally known at time t.

(ii) *Runge-Kutta methods:* The Runge-Kutta methods are based on computing x(t+δ) as a perturbation of x(t) by approximating the terms in a truncated Taylor series with comparable terms which do not involve derivatives. The Runge-Kutta methods are typically given as a means to approximating solutions to the class of differential equations defined by equation (18) (i.e., the special case of equation (1) where u(t)=t). Although this is not the class of primary interest in system theory, let us present the systems implications of the method in its normal context. We will consider modifying it later.

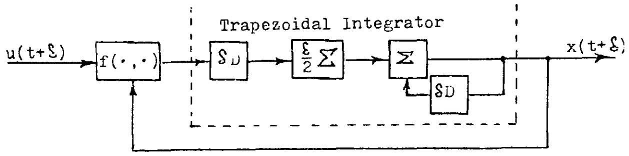


Figure 11—Discrete approximation to system of Figure 3 using a trapezoidal integrator

Perhaps the most widely used Runge-Kutta method is of order four:[4]

$$x(t+\delta)=x(t)+\frac{\delta}{6}(k_1+2k_2+2k_3+k_4) \tag{22}$$

where

$$\begin{aligned} k_1&=f(x(t),u(t)) \\ k_2&=f\left(x(t)+\frac{\delta}{2}k_1,\ t+\frac{\delta}{2}\right) \\ k_3&=f\left(x(t)+\frac{\delta}{2}k_2,\ t+\frac{\delta}{2}\right) \\ k_4&=f(x(t)+\delta k_3,\ t+\delta) \end{aligned} \tag{23}$$

Thus, in this Runge-Kutta method, integration is approximated as follows:

$$\int_t^{t+\delta} x(\tau)\, d\tau=\frac{\delta}{6}(k_1+2k_2+2k_3+k_4) \tag{24}$$

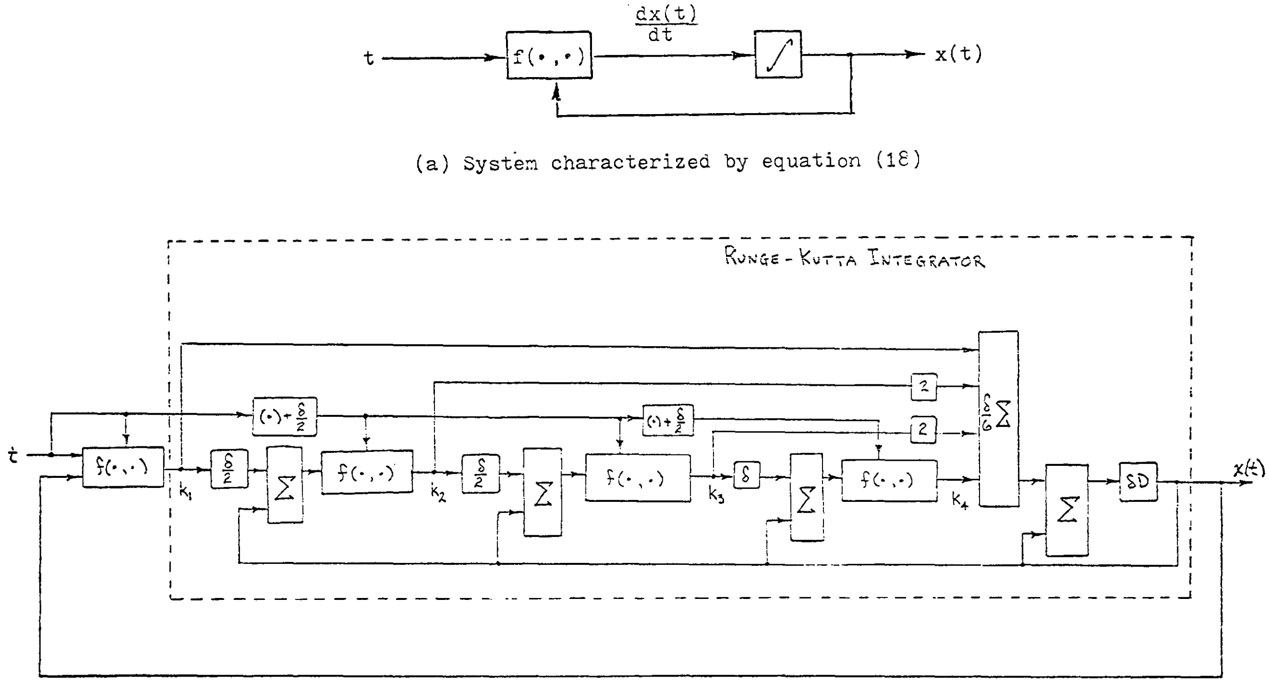
Figure 12 shows the original continuous system and the Runge-Kutta discrete approximation to it. It is noteworthy that:

(i) the Runge-Kutta integrator has only one delay element and no static loops or static through paths. Thus, only one initial value is required and, if the original continuous system contains no static loops, the resulting discrete system contains no static loops and an explicit set of difference equations results.

(ii) the Runge-Kutta integrator depends on the function f, unlike the Euler and trapezoidal integrators considered earlier. Thus, the Runge-Kutta integrator is *adaptive* in the sense that the integration process varies as a function of the signal being integrated.

(iii) the values of $t+\frac{\delta}{2}$ and t+δ are obtained in the system simply by adding $\frac{\delta}{2}$ and δ to t, respectively.

The point to the seemingly trivial observation (iii) is that static components are used to obtain future values of time. In particular, the computation of x(t+δ) $\left(\text{for the subsequent evaluation of } f\left(x(t)+\frac{\delta}{2}k_i,\ t+\frac{\delta}{2}\right),\ i=1,2, \text{ and } f(x(t)+\delta k_3,\ t+\delta)\right)$ does not require the knowledge of any function of time for values of time greater than t. In trying to use this Runge-Kutta method for finding the solutions of equation (1) (u(t)≠t), one is faced with evaluating $f\left(x(t)+\frac{\delta}{2}k_i,\ u\left(t+\frac{\delta}{2}\right)\right)$, i=1,2, and f(x(t)+δ, u



(a) System characterized by equation (18)

Figure 12—A continuous system and its Runge-Kutta discretization

(t+δ)). This means that at the time x(t+δ) is computed, the input u must be known for times $t+\frac{\delta}{2}$ and t+δ as well as for time t. Thus, if a real-time continuous system starts at time t=0, the corresponding real-time Runge-Kutta discrete system cannot start until time t=δ, at which time the values u(0), u $\frac{\delta}{2}$, and u(δ) are available, in addition to the original initial condition x(0).

IMPLICIT METHODS (THE PREDICTOR-CORRECTOR)

The class of predictor-corrector methods for solving differential equations provides an excellent example of how an implicit method can sometimes be utilized in numerical analysis. In this section the typical predictor-corrector is defined. The system-theoretic implications of the predictor-corrector integrator are noted. In particular, the predictor-corrector approach to the analysis of discrete systems with static loops is examined.

The predictor-corrector method is perhaps best introduced with respect to finding solutions of the no-input, time-invariant, one-variable system characterized by

$$\frac{dx(t)}{dt}=f(x(t)) \qquad (25)$$

Since some of the more accurate numerical integration methods, such as the trapezoidal method, are implicit methods (i.e., x(t+δ) is used in the computation of x(t+δ)), one may be forced to decide between computational expediency and computational accuracy. In spite of the computational difficulties associated with static loops encountered in using implicit methods, one may still decide to use them.

Most commonly used implicit methods can be represented by the following difference equation:

$$x(t+\delta)=x(t)+c(f(x(t+\delta)),f(x(t)),\ldots,f(x(t-k\delta))) \qquad (26)$$

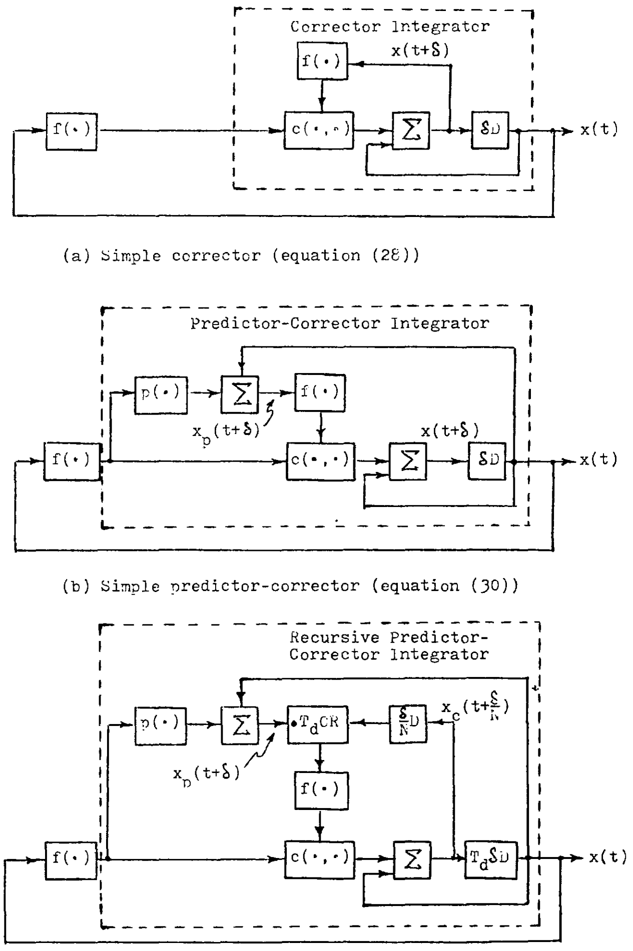
where x(t+δ), x(t), and the last k values of x are used to compute x(t+δ). Thus the function c provides the approximation to integration; it is called the *corrector*. If the trapezoidal method is used to solve equation (25), the result, in a form corresponding to that of equation (26), is

$$x(t+\delta)=x(t)+\frac{\delta}{2}(f(x(t+\delta))+f(x(t))) \qquad (27)$$

To simplify what follows, but with no loss of generality, consider a corrector which depends only on x(t+δ) and x(t) (e.g., that of the trapezoidal method):

$$x(t+\delta)=x(t)+c(f(x(t+\delta)),f(x(t))) \qquad (28)$$

Figure 13a shows a block diagram corresponding to the simple corrector system of equation (28). Note that a static loop exists in the corrector integrator.



(a) Simple corrector (equation (28))

(b) Simple predictor-corrector (equation (30))

(c) Simple recursive predictor-corrector (equation (33))

Figure 13—Corrector and predictor-corrector discretizations

The essence of the predictor-corrector method lies in the approach used to eliminate the static loop in the corrector integrator. An approximation to $x(t+\delta)$ is used in the corrector function c rather than $x(t+\delta)$ itself (which causes the static loop). The approximation to $x(t+\delta)$, $x_p(t+\delta)$, is obtained by using an explicit method requiring only $x(t)$ and the last k values of x:

$$x_p(t+\delta)=x(t)+p(f(x(t)),f(x(t-\delta)),\ldots,f(x(t-k\delta))) \qquad (29)$$

The function p used to anticipate the value of $x(t+\delta)$ for use in the corrector is called the *predictor*. For example, one might use the trapezoidal integrator as a corrector (equation (27)) and the Euler integrator $(x(t+\delta)=x(t)+\delta f(x(t)))$ as a predictor. Again, for simplicity, consider a predictor which depends only on $x(t)$ (e.g., that of the Euler method):

$$x_p(t+\delta)=x(t)+p(f(x(t))) \qquad (30)$$

Figure 13b shows a block diagram corresponding to the simple predictor-corrector system defined by equations (28) and (30). Note the predictor-corrector integrator has no static loops or static through paths.

For the case that a predictor-corrector method is to be used for a continuous system with an arbitrary input (i.e., equation (13)), the situation is again more complicated. Specifically, in real-time situations where a time-shifted system is not tolerable, one is faced with the problem of having to predict at time t the value of input $u(t+\delta)$, as well as the value of $x(t+\delta)$. In theory, since nothing is known about the exogenous system generating the input u, such a prediction is not possible. Practically, however, if the input u is not a totally random signal and δ is not too large, some method of extrapolation using past values of u might be used to predict $u(t+\delta)$.

The predictor-corrector configuration is also frequently used as the basis for an iterative scheme in which the predictor provides the first approximation to $x(t+\delta)$ and then the corrector, starting with the predictor's value, is used to iteratively generate a sequence of subsequent approximations to $x(t+\delta)$. The final value of this sequence of approximations to $x(t+\delta)$ is then taken to be the best approximation. The iterative scheme for generating the sequence of approximations to $x(t+\delta)$, for the simple predictor-corrector defined by equations (28) and (30), is as follows:

$$\begin{aligned}
x^{(1)}(t+\delta)&=x(t)+p(f(x(t)))\\
x^{(2)}(t+\delta)&=x(t)+c(f(x^{(1)}(t+\delta)),f(x(t)))\\
x^{(3)}(t+\delta)&=x(t)+c(f(x^{(2)}(t+\delta)),f(x(t)))\\
&\vdots \qquad\qquad\qquad\qquad\qquad\qquad (31)\\
x^{(N-1)}(t+\delta)&=x(t)+c(f(x^{(N-2)}(t+\delta)),f(x(t)))\\
x(t+\delta)&=x(t)+c(f(x^{(N-1)}(t+\delta)),f(x(t)))
\end{aligned}$$

The *lower-triangular form* of equations (31) is sufficient to assure that the system block diagram corresponding to this set of equations can be constructed so as to contain no static loops. However, an alternate representation, resulting in a significantly smaller block diagram, can be obtained by defining a new time set T_d' such that

$$T_d'=\left\{t_0, t_0+\frac{\delta}{N}, t_0+2\frac{\delta}{N}, \ldots, t_0+\delta, t_0+\delta+\frac{\delta}{N}, \ldots, t_f\right\} \qquad (32)$$

Note that $T_d \subset T_d'$. With respect to the new time set T_d' we write the following set of difference equations:

$$x_c\left(t+\frac{\delta}{N}\right)=x(t)+\begin{cases}c(f(x(t)+p(f(x(t)))),\\ \quad f(x(t))),\ t\epsilon T_d\\ c(f(x_c(t)),f(x(t))),\ t\notin T_d\end{cases}$$

$$(33)$$

$$x\left(t+\frac{\delta}{N}\right)=\begin{cases}c(f(x(t)+p(f(x(t)))),\\ \quad f(x(t))),\ t\epsilon T_d\\ x(t),\ t\notin T_d\end{cases}$$

where $x_c\left(t+\frac{\delta}{N}\right)$ is the corrector estimate of $x(t+\delta)$.

Figure 13c shows the system block diagram corresponding to equations (33). Note the subsequence $\{x(t_o), x(t_o+\delta), x(t_o+2\delta), \ldots, x(t_f)\}$, corresponding to the time set T_d, obtained from the output sequence of this system defined on time set T_d', is the same sequence one would obtain from the iterative predictor-corrector scheme defined by equations (31).

It is significant that an iterative scheme used to solve the nonlinear algebraic equations resulting from the existence of static loops in a discrete dynamic system corresponds to another dynamic system with no static loops. In essence, then, the static loop of the corrector system is eliminated by inserting a $\frac{\delta}{N}$ delay within it. To complete the iterative scheme, sufficient logic must be added to initiate the iterative scheme with the predictor estimate of $x(t+\delta)$ for every t such that $t \notin T_d$ and to assure that on this new time set, T_d', $x(t)$ changes only for times t such that $t \in T_d$. In Figure 13c this logic is conveniently achieved using the time-varying static element, the T_dOR, and the time-varying dynamic element, the T_d delay.

The T_dOR is a two-input (v_1 and v_2), single-output (v) static element defined such that if the *dot input* (i.e., the input associated with the arrow pointing to the dot) is v_1, then

$$v = \begin{cases} v_1, & t \notin T_d \\ v_2, & \text{otherwise} \end{cases}$$

In Figure 13c it is seen that a T_dOR is used to decide whethe̊r the predictor estimate of $x(t+\delta)$ or the corrector's own previous estimate of $x(t+\delta)$ is to be used in the corrector.

The T_d delay is a subsystem which is used to replace the δ delays in the original corrector system (defined on the time set T_d—Figure 13a). The T_d delay is, in effect, a δ delay defined on the new time set T_d'. The T_d delay with input v and output x is defined as follows:

$$x(t+\delta) = \begin{cases} v(t), & t \in T_d \\ x(t), & t \notin T_d \end{cases}$$

Figure 14 shows how a T_d delay can be realized using a T_dOR and two $\frac{\delta}{N}$ delays.

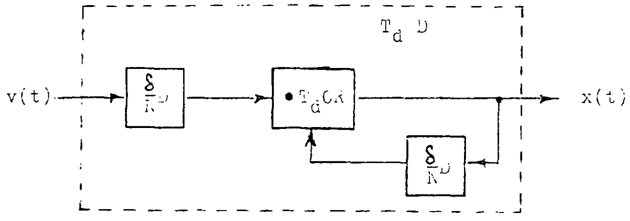


Figure 14—A realization of a T_d delay

Note there are three $\frac{\delta}{N}$ delays in the predictor-corrector recursive system shown in Figure 13c (two are within the T_d delay and thus not shown explicitly in the figure). The initial value of the $\frac{\delta}{N}$ delay within the T_d delay which provides the input to the T_dOR must be set to the same initial value given for the state variable x of the original continuous system (i.e., set to x_o); the initial values of the remaining two $\frac{\delta}{N}$ delays can be set to any values.

CONCLUSIONS

This paper presents a system-theoretic analysis of numerical methods used in approximating solutions of ordinary differential equations. By representing ordinary differential equations by system block diagrams (i.e., interconnections of static elements and integrators), a numerical method can be viewed as a process in which the integrators of a continuous system are replaced by discrete approximations to integrators (i.e., by discrete subsystems made up of interconnections of static elements and delays). Of special concern are questions concerning existence, uniqueness, and computability by digital computer of the solutions of the resulting difference equations which characterize the discrete system approximating the original continuous system.

A study is made of the properties of the system block diagrams characterizing canonical state-variable differential and difference equations. It is noted that proper interconnections of delays and static elements (i.e., discrete interconnections with no static loops) can always be characterized by explicit difference equations. Thus, a proper interconnection represents a computationally attractive form and as such is a goal in devising a numerical method. The main result of the paper establishes that if the system block diagram corresponding to the original differential equation has no static loops and if the discrete subsystems used to replace the integrators have no static loops and no static through paths, then the resulting discrete system has no static loops (and can thus be characterized by explicit difference equations).

With the main result in hand, a system-theoretic study is conducted of several of the more commonly used numerical methods: the Euler, trapezoidal, Runge-Kutta, and predictor-corrector methods. For each of these methods the discrete subsystem used to replace the integrators of the continuous system is detailed. The limitations of using these numerical methods in real-time analysis are examined, particularly with respect to the problem often encountered in which inputs must be anticipated. The system-theoretic implications of using the iterative predictor-corrector scheme are examined.

REFERENCES

1. Greenspan, Donald, *Discrete Models*, Addison-Wesley, 1973, pp. 5-10.
2. Windeknecht, T. G. and H. D'Angelo, "A System Graph and Canonical State Equations," *Proceedings Sixth Annual Southeastern Symposium on System Theory*, Baton Rouge, 1974.
3. D'Angelo, H. and T. G. Windeknecht, "Toward Computer Aids for Societal System Simulation," *Modeling and Simulation*, Vol. 5, Fifth Annual Pittsburgh Conference, 1974.
4. Daniel, James W. and Ramon E. Moore, *Computation and Theory in Ordinary Differential Equations*, W. H. Freeman, 1970, pp. 49-56.

Memory conserving efficient methods for solving large sets of stiff differential equations

by GRUIA-CATALIN ROMAN, DAVID GARFINKEL and CARL B. MARBACH
University of Pennsylvania
Philadelphia, Pennsylvania

ABSTRACT

Solution of large systems of stiff differential equations by the most widely used method, that of Gear, is limited by its requirement for the presence in memory of a Jacobian matrix, which may become intolerably large. Methods of alleviating this situation, believed to be broadly applicable, have been worked out with large metabolic models. One general tactic is to remove from the set of differential equations elements which can be represented instead by algebraic equations: rate equations for enzymes and equilibrium relations for very rapid reactions. Another general tactic is to store efficiently only a part of the sparse Jacobian matrix containing the non-zero elements, which requires the presence of a preprocessor, using algorithms which are given in the paper. The Jacobian matrix may sometimes be represented by a diagonal approximation; this works better with the algebraic methods. Examples of the savings are given. These methods have been incorporated into BIOSSIM, a machine-independent simulation language designed for large biological systems, and in our hands have considerably reduced the cost and difficulty of solving large systems of stiff differential equations.

Methods of solving "stiff" differential equations have received considerable attention within the last few years, and important improvements have recently been made.[1] The most important of these is the predictor-corrector method of Gear.[2] This appears to be the most widely used "stiff" differential equation solver, and is widely considered to be the best one. This implicit method requires that the Jacobian matrix of the differential equation variables be present in core memory. Unfortunately the size of this matrix can become intolerably large when modeling complex systems. We describe here an economical method of computing the behavior of large systems based largely on modifications of Gear's method, which we have thus far applied to biochemical systems. The most important such modification requires the presence in memory only of the non-zero elements of the Jacobian matrix, usually a small part of the total for the systems we have worked with.

Biological systems are often inherently quite complicated. Accurate representation of such systems in terms of differential equations ordinarily results in "stiffness"; this is widely believed to hold true for natural systems generally. A common problem in simulating them is that simplification for the sake of mathematical tractability often leads to biologically unrealistic results. Representing a system in terms of n differential equations (where often $n > 100$) may lead to the following serious dilemma:

(1) n must be large enough for biological realism;
(2) n^2 (the size of the Jacobian matrix) must be small enough for available core memory.

The methods we have developed reduce core memory requirements to a practical range and also save computer time. While they are based on the structure of the biological systems which we have investigated, the methods can be extended in varying degree to other complex systems of stiff differential equations.

We have used two main approaches:

(1) Replace some of the differential equations with algebraic equivalents when this is possible.
(2) Reduce the number of elements of the Jacobian matrix required to be stored in memory for the Gear method to solve the remaining differential equations.

While these two approaches can be combined for a given system, as exemplified later, they are described separately in the two following sections.

SUBSTITUTION OF ALGEBRAIC RELATIONSHIPS

The general approach employed is to separate out from the total system those components whose behavior can be calculated by algebraic equations rather than differential equations, thus greatly reducing the number and possibly the "stiffness" of the latter.

In metabolic models enzyme mechanisms and reactions at equilibrium can be separated out and dealt with by specialized algebraic means:

(1) Individual enzyme behavior in a complex model can be represented by an algebraic rate law[3] which can be determined by standard systems analysis procedures.[4] The differential equations for these enzyme forms are converted to explicit algebraic equations. The concentrations of the enzyme forms and the rate at which the enzyme reactions go are calculated by matrix inversion.[5] Only the small molecules involved need be represented by differential equations. The saving of CPU time and memory is indicated by the last two columns of Table 1, although a little space is required for the algebraic equations.

(2) Some chemical reactions (e.g., inorganic chelations) are very rapid compared to others in the model and can be represented as always being at equilibrium. Techniques for calculating these equilibria, which have existed for some time[6] have been adapted for joint use with differential equations solving[7,8] and incorporated into the program described here. As these reactions (here referred to as "fast reactions") are the ones with the fastest time constants, removing them from the system of differential equations reduces the "stiffness".

The differential equations which remain after the above elements have been separated out are then solved by the Gear method. The efficiency of Gear's differential equation solver is independent of stiffness over a wide range of stiffness, although we have been able to slow it down by making the equations very stiff. However, if a diagonalized approximation is used[9, 10] for the Jacobian matrix in Gear's method, the sensitivity to stiffness appears to increase considerably. Separating out the fast reactions alleviates this situation by reducing the stiffness. The diagonalized approximation may also be aided by the fact that the elements removed from the Jacobian matrix in this process are commonly off-diagonal elements as well as being the (absolutely) largest ones.

MODIFICATION OF THE GEAR METHOD

In the course of considerable experimentation with various versions of the Gear method we found it to be more than satisfactory. We made efforts to improve its performance by reducing its execution time as well as the space requirements. These changes can be considered "tuning" of the Gear method, which speeded things by 10-15 percent. Analysis of the program's behavior revealed that a large percentage of the differential equation solving time is spent in solving a linear system of equations of the form:

$$X * J = B \tag{1}$$

where J is the Jacobian or partial derivative matrix mentioned above. Furthermore the *size* of the J is the factor determining the time spent in finding the solution for X, and reducing the size of J will reduce execution time as well as space requirements.

It is a fortunate situation that the Jacobian matrix for biological models representing metabolic systems is normally quite sparse, so that sparse-matrix techniques are applicable. These were applied by Curtis and Chance in the CHECK and CHECKMAT program.[11] It is possible to go further because the sparseness is structured so that most of the non-zero elements of the matrix are near the main diagonal, as shown in Figure 1. Furthermore, the non-zero elements which are not near the main diagonal may also have a structure which can be exploited; this is in fact done by the removal of fast reactions described in the preceding section. It would be desirable to eliminate the zero elements of the sparse Jacobian matrix from storage in the most efficient way. However, this could not be achieved initially due to the method used by Gear to solve the linear system—backward decomposition with partial pivoting. The pivoting is dependent upon the value of the elements of J at each given point. This makes it impossible to predict how the sparseness of J is changed by the decomposition algorithm. On the other hand, we discovered that the partial pivoting made little difference in the behavior of the Gear's method and could be replaced by a simpler backward decomposition procedure. Once pivoting is eliminated, predicting the effect of the decomposition upon the sparseness of J becomes possible.

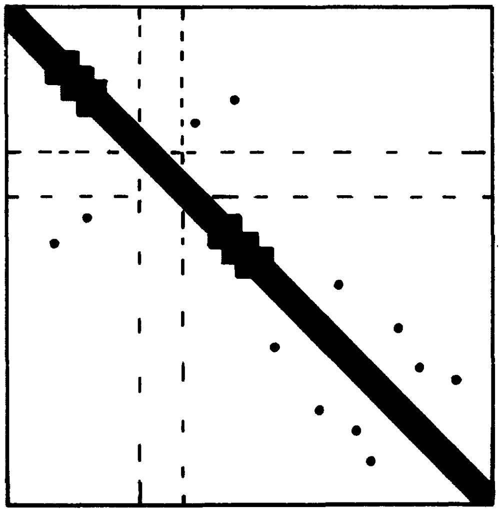


Figure 1—Typical sparseness pattern of the Jacobian matrix representing a metabolic system (non-zero elements are indicated in black)

PROCEDURE FOR SPARSE MATRICES

The sparseness prediction algorithm is:

1. $i=1$ (*the sparseness of the first row will never be altered* by the decomposition process).
2. $i=i+1$ and $j=1$ (consider the next row).
3. if $i>N$ go to step 9 (if all N rows of the Jacobian J have been analyzed, exit the loop).
4. increment j until the next non-zero element is encountered (all zero elements skipped so far on the row i will always remain zero).
5. if $i=j$ go to step 2, (*the sparseness of row i was determined*).
6. if $j>i$ stop—the matrix J is not a Jacobian.
7. all elements $J(i, t)$ for which $t>j$ and $J(j, t) \neq 0$ will be marked as non-zero, unless they are already marked as such.
8. go to step 4.
9. the algorithm stops:
   all elements of J which have not been marked as non-zero will in no circumstances become non-zero as a result of the decomposition process. The decomposition algorithm will be described below.

Example: application of the sparseness prediction algorithm to a sample Jacobian. In the following, '*' will indicate original non-zero elements, while a '+' will signify that the given element was marked as non-zero during the application of the algorithm. All elements left unmarked by the sparseness prediction algorithm need not be present in core during the integration process.

row 1	row 2	row 3	row 4	row 5
*0*0*	*0*0*	*0*0*	*0*0*	*0*0*
	0*0*0	0*0*0	0*0*0	0*0*0
0*0*0		*0*0+	*0*0+	*0*0+
*0*00	*0*00		00**+	00**+
00**0	00**0	00**0		0*0+*
0*00*	0*00*	0*00*	0*00*	

Subsequently, the sparseness prediction algorithm is used to create two arrays WI and WJ. For each non-zero element in J an entry (indicating the column number of the non-zero element) is made in WJ. All entries in WJ are ordered by rows and the elements in each row ordered in increasing order (by column numbers). Furthermore, for each row a pointer in array WI is set to indicate its beginning (e.g., the column of the first non-zero element of row i is given by WJ(WI(i))). An additional pointer binds the last row. In the case of the Jacobian used in the previous example, the arrays WI and WJ take the following form:

WI: 1 4 6 9 12 15
WJ: 135 24 135 345 245

The preprocessor will pass to the simulator not only the dimensions of WI, WJ, and WPW (the Jacobian dimensioned to the same size as WJ), but also the content of WI and WJ.

The saving of memory space resulting from this process becomes more significant in the case of our implementation, described below, which is composed of two programs: a preprocessor (called the generator), and the simulator program which actually solves the differential equations. The generator receives as input the set of chemical reactions to be represented by a set of differential equations and, on analyzing it, dimensions the simulator's arrays to the exact length required by the specific system. It then derives the set of differential equations themselves, following which the task of determining the initial sparseness of the Jacobian becomes trivial.

Whenever required, the simulator will compute the values of the Jacobian elements and store them in the appropriate locations of WPW. The following decomposition routine will replace WPW by a triangular matrix (stored in the upper corner of the Jacobian) and a triangular multiplication factor matrix (stored in the lower side of the Jacobian). The decomposition algorithm is quite similar to the sparseness prediction algorithm since the latter mimicks the actions of the former (N is the number of variables involved; J is the conceptual Jacobian, not to be identified with its program implementation WPW):

1. $i=1$ (the first row is unchanged).
2. $i=i+1$ and $p=0$ (consider the next row).
3. if $i>N$ go to step 9 (the triangularization is complete).
4. find next non-zero element on row i: $J(i, j) \neq 0$, i.e., repeat $j=WJ(WI(i+p))$ and $p=p+1$ until $WPW(WI(i+p)) \neq 0$ (note that j steps over the elements which are known to be zero).
5. if $i=j$ go to step 2.
6. if $j>i$ stop—programming error.
7. (a) set $J(i,j)=K$ (i.e., $WPW\ (WI(i+p))=K$) where $K=-J(i,j)/J(j,j)$.
   (b) add row j to row i starting with the column $j+1$.
8. go to step 4.
9. stop—algorithm is completed.

By using the result of this decomposition routine, the solution routine will compute the value of the unknowns of the equation (1) for each given B.

In accordance with our expectation, the predictive sparseness method resulted not only in substantial savings in core but also in a significant improvement in the speed of Gear's method: the time required to solve the differential equation system decreased considerably. The advantages resulting from this method become more important with increasing system size. To our knowledge, the only other successful research of this type resulted in a (DEC-10) machine-dependent program (M. Pring, to be published) which is not yet available for consultation.

The predictive sparseness method is the last of a large series of experimental methods which unfortunately could not pass our acceptance criteria—they had to prove themselves faster than the regular Gear method. In our first attempt, we tried to use a variation of Gear's method where the Jacobian is replaced by a diagonal approximation, and this was excessively slow for large stiff systems and suffered a loss in accuracy when the same relative error parameter was used. We also tried a new version of Gear's package, GEARB[10] which replaced the Jacobian by a diagonal band of chosen width. Since most of our systems involve Jacobians which are concentrated around the diagonal (Figure 1), the chances of success seemed good. Furthermore, our experiments showed improvements in the execution speed when the Jacobian indeed had all the non-zero elements inside such a band. However, the method was rejected on further testing, which revealed that as soon as elements outside the band became slightly significant the behavior of the program is adversely affected.

A second group of experiments followed in an attempt to use a Gauss-Seidel iteration procedure to solve the set of linear equations (1). In the belief that the slowdown encountered in the previous cases was due to a poor approximation of the Jacobian, we attempted to use the solution given by the diagonal band as a starting point for the iteration process. However, the convergence was much too slow, especially for large systems.

POSSIBLE ALTERNATIVE METHODS

As mentioned before, the predictive sparseness method gives good results and the larger the number of differential equations, the greater the improvements appear to be. However, alternative differential equation solving methods are available to the user of our program: Euler, regular Gear, or Gear with the Jacobian replaced by a diagonal approximation. All of these methods may also be combined with the fast reaction method mentioned above.

The Euler method is rarely used to integrate stiff differential equations, but past experience has shown it to work as well—or as badly—as the more sophisticated non-stiff methods when faced with a stiff system. Since decreasing stiffness improves the behavior of Euler, it may sometimes be used in conjunction with the fast reaction method; if the latter greatly reduces the degree of stiffness, this combined method will give an accurate solution in a reasonable time for systems where Euler alone would fail. Since the Gear method is "slow-starting" the Euler method is at a relative advantage for short-time calculations, as seen in the first two columns of Table 1 (the second column includes editing time, the first does not).

TABLE 1

Computing times to solve three different test systems on the PDP-10 computer. Except for the first column, these include a considerable computer expenditure for graphing and tabulating the solutions. The programs involved are all in FORTRAN except for the ones to algebraically handle enzymes, which are in assembly language (here the advantage in speed over FORTRAN is particularly great). J is Jacobian size (unaffected by fast reactions).

Test Systems

Size	N=9		N=65	N=66
Description	No fast reactions, no enzymes, not very stiff		No fast reactions, not very stiff, enzymes represented by differential equations	Fast reactions, enzymes represented algebraically
Method of Solution	time=1500	time=15		
Euler with fast reactions	29.3 sec J=0	15.5 sec J=0		67.9 sec J=0
Gear diagonal approximation	26.1 sec J=9	15.9 sec J=9		unworkable —
with fast reactions				18.6 sec J=20
Regular Gear	8.8 sec J=81	24.5 sec J=81	12 min 25.9 sec J=4096	22.2 sec J=400
with fast reactions				41.4 sec J=400
Gear-predictive sparseness	6.6 sec J=34	20.8 sec J=34	6 min 43.3 sec J=1715	17.0 sec J=47
with fast reactions				40.8 sec J=47

The diagonal approximation of the Jacobian suffers from many of the same weaknesses as Euler, but to a lesser degree. However, when *properly* combined with the fast reaction method, surprisingly enough, it is faster than the regular Gear method, for some medium-sized systems tested. This is very significant when we think that only N locations are needed for the Jacobian in place of N*N. This method is still under investigation, but we do not hope for equally good results with larger systems. Moreover, the fast reaction method has applicability restrictions (e.g., it requires a clean distinction between fast and non-fast reactions) which make it unsuitable in many cases.

The predictive sparseness method will not work where the user interferes with the derivative evaluation routine (written by the preprocessor) by inserting additional (FORTRAN) coding and artificially creating partial derivatives of significant size, which cannot be predicted since they are not derived from the original differential equations. The unmodified Gear method does work in this situation.

A short comparative study of the various methods available in the system is summarized in Table 1 (CPU time for integration and the size of core used for the Jacobian).

DEPENDENCE ON WORDLENGTH

During our investigations[12] we were able to run our program on three different machines, the Control Data 6400, the IBM 370/165, and the Digital Equipment PDP-10 (with which the bulk of the work was done). We were investigating the effect of wordlength of the accuracy and number of calculations required to solve a test problem. As expected, the 32 bit processor (IBM) gave slightly less precision and performed more calculations in reaching a solution than did the PDP-10. Surprisingly the 6400 (60 bits) performed even more calculations than did the 32 bit IBM 370. Further investigation showed that a minimum number of calculations occurred at wordlengths of 36 to 40 bits.

Since the Jacobian is formed by numerical differencing within the Gear program, increased wordlength will lead to some elements being very small but finite where shorter wordlength would result in such elements being set to zero. If too many are set to zero, accuracy can be lost in forming the Jacobian, thus requiring more evaluations for convergence. If too few are set to zero, there should be no problem. The pivoting strategy used in the matrix decomposition routine in the original Gear program employed no scaling, so that these small elements affected the pivot with a similar loss in accuracy causing additional matrix evaluations. An earlier version with scaling did not demonstrate this phenomenon. Since our latest method does not employ pivoting, the problem has been eliminated. This is mentioned here because this result is so unexpected: increased wordlength is generally believed to improve the efficiency of matrix computations.

IMPLEMENTATION IN A SIMULATION LANGUAGE: BIOSSIM

Application of these methods of handling the Gear matrix necessarily requires a preprocessor of some kind to determine what non-zero derivatives are possible in the Jacobian matrix and to construct pointers for them. We have combined the Gear method into a machine-independent two-pass language biochemical simulation language which we have been using for some time.[13] This language which writes and solves differential equations can perform the necessary operations in its first pass. Our research into methods for solving stiff differential equations grew out of the need to make this language more efficient. It has otherwise been updated considerably, and a report of this will be submitted elsewhere. A most important aspect of this updating is that the program implementing the language have been "structured" to permit modification by the user.[14] Given the capability of such modification and the ability to write as well as solve ordinary differential equations, this language can probably be applied to systems of equations far removed from those for which it was designed. It has already been used for problems which are more nearly physiological than chemical in their definition and for ecological problems. As communication about such usages would be considerably facilitated by having an acronym for the language, we have named it BIOSSIM (for BIOlogical Structured SIMulator).

The ability to solve large numbers of stiff differential equations is expected to considerably assist the study of biological systems by mathematical means, because the equations can be sufficiently complex to meet the need for realism. It has been difficult to develop such computer models in the past because of the cost of computer time, and the unfavorable running restrictions (e.g., nighttime access only, because of the large core requirement). There is reason to hope that the biological models that are feasible to compute with the methods here described will be realistic enough to be of value for clinical and industrial applications.

The BIOSSIM program will be available, as its predecessors for some time have been, through the SHARE Program Library Agency (Library No. 360D-03.2.008), and possibly in other ways as well.

SUMMARY

A method of decreasing both the running time and the core occupancy of the Gear stiff differential equation solver by "compacting" its large Jacobian matrix with a preprocessor and auxiliary programs is described. This may be further assisted by separating out appropriate subelements of a large set of stiff differential equations and treating them by algebraic methods instead. In our hands these methods have had a very large impact on the difficulty and cost of solving complex systems of stiff differential equations.

This project was supported by NIH grants GM 16501, HL 15622, and RR 15.

REFERENCES

1. Willoughby, R., ed., *Stiff Differential Equations*, New York, Plenum Press, 1974.
2. Gear, C. W., "The Automatic Integration of Ordinary Differential Equations," *Comm. ACM 14*, pp. 176-9, 1971.
3. Rhoads, D. G., M. J. Achs, L. Peterson, and D. Garfinkel, "A Method of Calculating Time-Course Behavior of Multi-Enzyme Systems from the Enzymatic Rate Equations," *Comput Biomed Res 2*, pp. 45-50, 1968.
4. Garfinkel, L., M. C. Kohn and D. Garfinkel, "Systems Analysis in Enzyme Kinetics," *CRC Crit Rev Bioeng*, in press.
5. Rhoads, D. G. and M. Pring, "The Simulation and Analysis by Digital Computer of Biochemical Systems in Terms of Kinetic Models," *J Theor Biol 20*, pp. 297-313, 1967.
6. Deland, E. C., *Chemist—The Rand Chemical Equilibrium*

Program, Memo RM-5404-PR, Rand Corp., Santa Monica, Calif., 1967.

7. Clasen, R. J., *The Solution of Chemical Kinetics Problems that Produce Stiff Differential Equations,* PhD Thesis, University of California, Los Angeles, 1974.
8. Clasen, R. J., D. Garfinkel, N. Z. Shapiro and G.-C. Roman, "A Method for Solving Certain Stiff Differential Equations," *SIAM Journal on Appl Math,* submitted for publication.
9. Hindmarsh, A. C., *Linear Multi-step Methods for Ordinary Differential Equations: Method Formulations, Stability, and the Methods of Nordsieck and Gear,* Memo UCRL-51186 Rev., Lawrence Livermore Laboratory, University of California, Livermore, 1972.
10. Hindmarsh, A. C., *GEARB: Solution of Ordinary Differential Equations Having Banded Jacobian,* Memo UCID-30059 Rev. 1, Lawrence Livermore Laboratory, University of California, Livermore, 1975.
11. Curtis, A. R. and E. M. Chance, *CHEK and CHEKMAT: Two Chemical Reaction Kinetics Programs* AERE-R 7345, United Kingdom Atomic Energy Authority, Harwell, Oxfordshire, 1974.
12. Marbach, C. B., *Differential Equation Solving in Biochemical Simulation: A New Method and Implementation,* M S Thesis, University of Pennsylvania, 1973.
13. Garfinkel, D., *A Machine-Independent Language for the Simulation of Complex Chemical and Biochemical Systems, Comput. Biomed Res 2,* pp. 31-44, 1968.
14. Roman, G.-C., *Program Control Restructuring as a Methodology for Systematic Programming,* Ph D Thesis, University of Pennsylvania, 1976.

A geometric analysis of heuristic search

by GORDON J. VANDERBRUG
University of Maryland
College Park, Maryland

ABSTRACT

Search spaces for various types of problem representations can be represented in one quadrant of the coordinate planes. This geometric representation is used to prove some formal properties of heuristic search strategies involving completeness, admissibility, optimality, consistency, and the use of the perfect heuristic. The geometric analysis provides an intuitive alternative to the algebraic analysis which appears in the literature.

INTRODUCTION

To specify a state-space representation for a problem, one must specify a start state, the general structure of a state, a characterization of a goal state, and a set of operators which map states into states. To solve a problem using such a representation one successively applies operators to currently generated states to obtain new states until a goal state is generated. This process is called searching the search space (or state-space), and can be done in many different ways. The manner in which it is done is called a search strategy. The solution to the problem is the sequence of operators which transforms the start state into the generated goal state.

The graph model for searching a state-space is based on associating states in the representation with nodes of the graph, and operators of the representation with arcs of the graph. With this model we can view a state-space representation as an implicit definition of a graph. It defines a start node and procedures for generating other nodes of the graph.

A search strategy can be thought of as a process of making explicit part of this implicitly defined graph. In this paper we will be dealing with a frequently occurring class of search strategies called ordered search strategies. An ordered search strategy determines the manner in which the state-space is generated by assigning a merit ordering to the nodes. A merit ordering is a procedure for ranking the nodes. The search process occurs in stages, and at each state the merit ordering specifies which node is to be expanded (i.e., which node is to have its successor nodes generated by applying all applicable operators). An ordered search algorithm is presented in Figure 1. It is equivalent to the algorithms presented in Hart, Nilsson, and Raphael;[1] Pohl;[2] and Kowalski.[3]

Usually merit orderings are defined by evaluation functions. Evaluation functions use selected features of a state to assign it a number, and thus rank it relative to all the other states. One possible evaluation function assumes that the operators have associated costs, and assigns to each node the sum of the costs of the operators used to generate it. This evaluation function is usually denoted by g, and the search resulting from its use is called the uniform cost search strategy. In terms of the graph which represents the

```
Set $S$ to the empty set and $\tilde{S}$ to the start node
*While* $\tilde{S}$ is not empty *do*
  Choose $n \in \tilde{S}$ such that n has best merit (resolve ties arbitrarily)
  *If* n is a goal node
    *then*
      Exit with success
    *else*
      Place n in S
      *For* each $m \in \Gamma(n)$ *do*
        *Cases*
        1) $m \notin S$ and $m \notin \tilde{S}$
             place m in $\tilde{S}$ with pointer to n
        2) $m \in \tilde{S}$ and new merit is better than old merit
             set pointer to n and redefine merit
        3) $m \in S$ and new merit is better than old merit
             place m in $\tilde{S}$ with pointer to n and redefine merit
  *end*
  Exit with failure
```

Figure 1—An ordered search algorithm. S is the set of nodes which have been expanded, $\tilde{S}$ the set of nodes which are candidates for expansion, and $\Gamma(n)$ the finite set of successors of the node n. The algorithm forms a tree of the nodes in the problem space, with the nodes in $\tilde{S}$ at the tips of the tree. When the algorithm exits with success the solution path can be found by tracing the pointers from the goal node to the start node

state-space, the uniform cost strategy uses the cost of the path from the start node to the node n as the merit of n. Pure heuristic search uses and estimates, h(n), of the cost of a path from n to the nearest goal node as the merit of n. Frequently h(n) is based on the extent to which selected features of the node n differ from these same features of a goal node. This cost is frequently referred to as the distance from n to a goal. We will insist that $h(n)=0$ whenever n is a goal node. An ordered search strategy which uses an evaluation function $f(n)=g(n)+h(n)$, with both a cost and a heuristic component, is called diagonal search. Various weights on the cost and heuristic components can be achieved with the function $f(n)=(1-\omega)g(n)+\omega h(n)$, for $\omega\epsilon[0,1]$. $\omega=0$ and $\omega=1$ are uniform cost and pure heuristic search respectively; while $\omega=\frac{1}{2}$ is diagonal search, since the two evaluation functions, $\frac{1}{2}g(n)+\frac{1}{2}h(n)$ and $g(n)+h(n)$, define the same merit ordering.

DEFINITIONS

An ordered search strategy is said to be complete if whenever there exists a solution to the problem the strategy will find one. An admissible strategy is one which terminates with a minimum cost solution whenever one exists. The concept of optimality applies to strategies which are admissible and is defined as follows. Let h_1 and h_2 be two heuristic functions such that $h_2(n)<h_1(n)\leq h_p(n)$ for all nongoal nodes n, where h_p is a perfect heuristic function (the heuristic which gives the exact cost to the nearest goal). An admissible strategy is said to be optimal if searching with h_2 expands all of the nodes that searching with h_1 expands. Admissibility can be viewed as the optimality of the solution, whereas optimality is really the optimality of the search process.

In order to prove that ordered search strategies possess the above properties certain assumptions on the heuristic function and the graph of the search space must be made. A heuristic h satisfies the lower bound condition if $h(n)\leq h_p(n)$ for all n, where h_p is the perfect heuristic. A heuristic h is said to be consistent if any n and n′ such that there is a path from n to n′, $h(n)-h(n')\leq k(n,n')$, where k(n,n′) is the cost of the path from n to n′. The concept of a δ-graph will also be used in the theorems. We will define a δ-graph to be a graph which does not contain a path with an unbounded sequence of partial sums of arc costs.

GEOMETRIC REPRESENTATION

The proofs of the theorems in this section are geometrically-based proofs, in that they use a method of representing the search space in one quadrant of the coordinate plane. Each node n has a cost g(n) and a heuristic h(n) associated with it (for uniform cost search $h\equiv0$). Thus we can represent the node n at the point (i,j), where $i=h(n)$ and $j=g(n)$. This representation maps the entire search space into one quadrant of the plane (many nodes can be mapped into the same point, but this is of no consequence), and gives us the basis for geometrically describing the search process.

Figure 2 provides an example of the geometric representation of a search space. Note that the node n is represented at the point (h(n), g(n)), and that the h-axis extends horizontally to the right while the g-axis extends vertically downward. Since the g-component of any start node is zero, all start nodes lie on the h-axis. Similarly, all goal nodes lie on the g-axis since their h-components are zero. The nodes n_6 and n_7 could be placed in more than one position in the plane since there is more than one path from n_0 to each. In Figure 2 we have chosen the position which corresponds to the shortest path.

The heuristic used in Figure 2 satisfies the lower bound condition since it can be verified that the $h(n_i)$ is less than or equal to the distance between n_i and n_7 for $i=1,...,7$. However, this heuristic is not con-

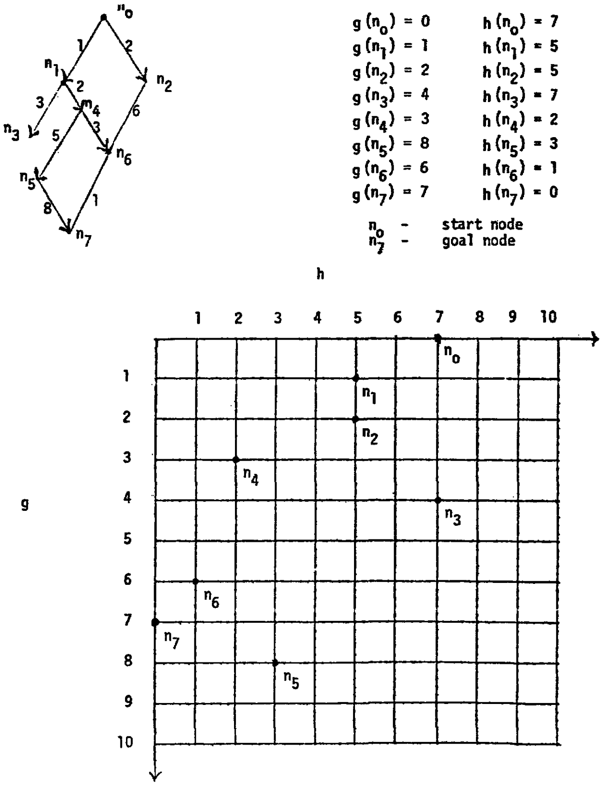

n_0
1
2
n_1
n_2
2
3
n_4
6
n_3
5
3
n_6
n_5
8
1
n_7
$g(n_0) = 0$ $h(n_0) = 7$
$g(n_1) = 1$ $h(n_1) = 5$
$g(n_2) = 2$ $h(n_2) = 5$
$g(n_3) = 4$ $h(n_3) = 7$
$g(n_4) = 3$ $h(n_4) = 2$
$g(n_5) = 8$ $h(n_5) = 3$
$g(n_6) = 6$ $h(n_6) = 1$
$g(n_7) = 7$ $h(n_7) = 0$
n_0 - start node
n_7 - goal node
h
1 2 3 4 5 6 7 8 9 10
g
1 2 3 4 5 6 7 8 9 10
n_0
n_1
n_2
n_4
n_3
n_6
n_7
n_5

Figure 2—An example of the coordinate representation of a search space

sistent since (for example) $h(n_1)-h(n_4)=5-2=3$, while $k(n_1,n_4)=2$.

Each choice of ω in $f_\omega=(1-\omega)g+\omega h$ determines the direction in which the space will be searched. Search with $\omega=\frac{1}{2}$ is called diagonal search because it defines all nodes which lie on the same diagonal to have equal merit and attempts to expand nodes in the direction indicated in Figure 3(a). Upwards diagonal search differs from diagonal in that node n has better merit than a node n′ iff $f(n)<f(n')$, or $h(n)<h(n')$ when $f(n)=f(n')$; and attempts to expand nodes in the same direction as diagonal search except that the search proceeds up successive diagonals as indicated in Figure 3(b). Figures 3(c)-3(f) illustrate the direction of search for various values of ω. If the distinction between the h-components is made for nodes with the same f-value as is done in upwards diagonal search, then along each line in these figures the search proceeds in the direction of the h-axis.

As the process of searching the space proceeds, the region of the quadrant which has been covered grows. As any stage of the search process all nodes which lie inside the covered region (including the boundary) can potentially be expanded. A node n lying inside such a region will be expanded if there is a path from the start node to n such that all the nodes on this path also lie within the region. Note that n lying within the region covered by the search is not sufficient reason for it to be expanded. Note also that a node which is expanded does not necessarily lie on the diagonal of the triangular region, but may lie strictly within the region.

An alternative to specifying an evaluation function in the form $f_\omega=(1-\omega)g+\omega h$ for $\omega\epsilon[0,1]$ is to use the form $f_\alpha=g+\alpha h$ for $\alpha\epsilon\ [0,\infty)$. Since scaling the evaluation function does not change the merit ordering defined by it, the relationship between these two forms is given by $\alpha=\omega/(1-\omega)$. For example, diagonal search is specified by setting either $\omega=\frac{1}{2}$ or $\alpha=1$ in these respective formulas.

When the first form is used, changes in the value of ω correspond most naturally to changing the direction of search as illustrated in Figure 3. However, the parameter α in the second form can be thought of as being part of the heuristic component of the evaluation function. When this is done, it is most natural to think in terms of the direction of search remaining diagonal, but all the nodes being moved either towards the g-axis if $\alpha<1$, or away from the g-axis if $\alpha>1$. Thus the above two forms point out the two ways in

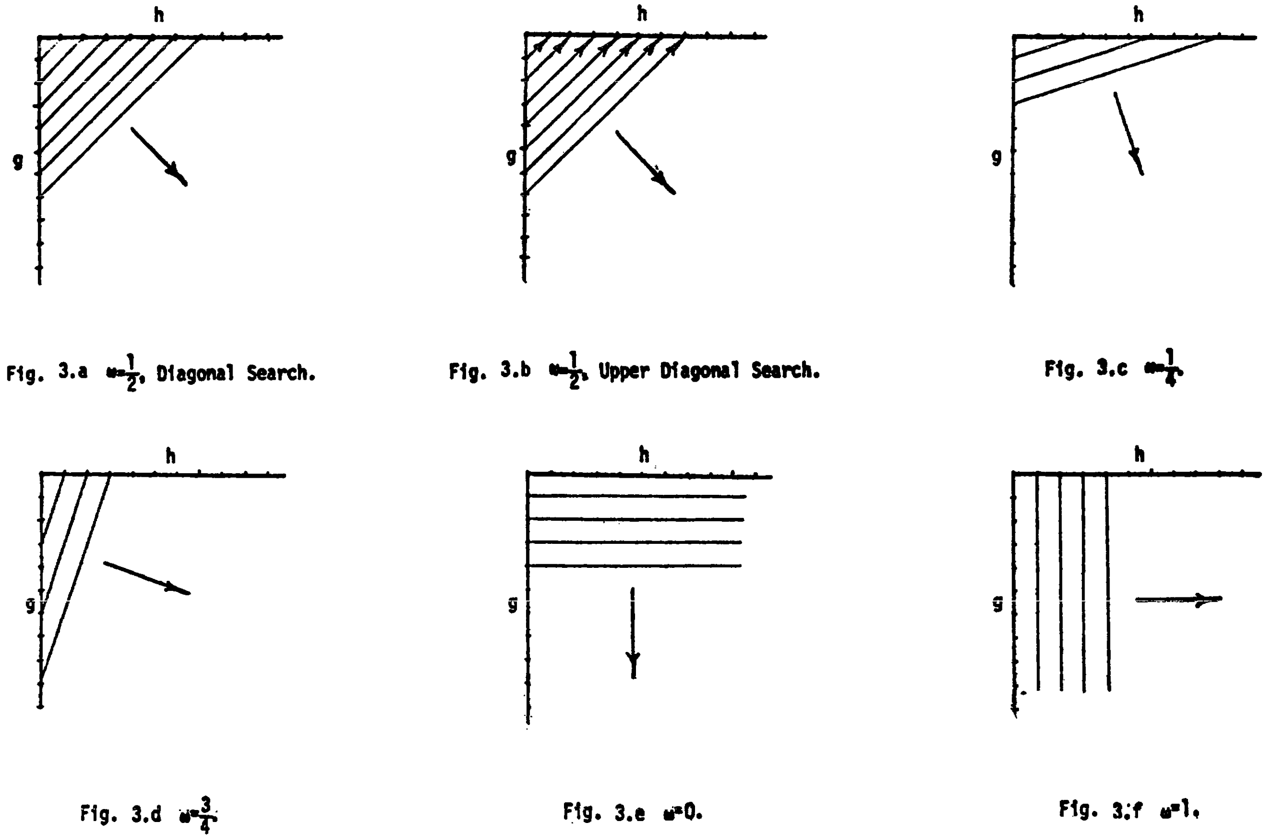


Fig. 3.a $\omega=\frac{1}{2}$, Diagonal Search.

Fig. 3.b $\omega=\frac{1}{2}$, Upper Diagonal Search.

Fig. 3.c $\omega=\frac{1}{4}$.

Fig. 3.d $\omega=\frac{3}{4}$.

Fig. 3.e $\omega=0$.

Fig. 3.f $\omega=1$.

Figure 3—The direction of search in the coordinate representation of the search space for various values of ω

which a change in an evaluation function can be viewed; either as a change in the direction of search or as a change in the position of the nodes. In this paper we will use the $f_\omega = (1-\omega)g+\omega h$ alternative.

THEOREMS

Now that we have explained the geometric representation of the search space, we turn to the statements and proofs of the theorems. The proofs of the theorems refer to the illustrations in Figure 4. The term search strategy will mean an ordered search strategy which uses the evaluation function $f_\omega = (1-\omega)g+\omega h$.

THEOREM 1. [Completeness] A search strategy is complete for all δ-graphs iff $\omega \epsilon [0,1)$.

PROOF: Assume that there is a solution, that is, a path from the start node s to a goal node t, and that $\omega \epsilon [0,1)$. Thus, the direction of search is not parallel to the g-axis (the case where $\omega = 1$). This case is illustrated in Figure 4(a). Suppose that the solution is not found. Then the search strategy never reached all of the nodes on the solution path because some region in the plane encountered by the search strategy contains an infinite number of nodes. Since $\omega \neq 1$, such a region must be either an infinite strip parallel to the h-axis (the case where $\omega = 0$), or an finite triangular region (the case where $0 < \omega < 1$). In either case, since some region contains an infinite number of nodes, there exists a path in the graph whose g-values are unbounded. A path of unbounded g-values is the only condition which can give rise to an infinite number of nodes in either a region parallel to the h-axis or a finite triangular region. However, a path of unbounded g-values is impossible because the graph is δ-finite. Hence, if $\omega \epsilon [0,1)$ then all the nodes along the path from s to t will eventually be expanded; that is the search strategy is complete.

Now suppose that $\omega = 1$. Then since there can be an infinite path in the graph whose h-values are bounded above, one of the infinite strips which are encountered by the search strategy before it covers all of the nodes from s to t can contain infinitely many nodes. Such a strip would prevent the search strategy from finding the solution. Thus search with $\omega = 1$ is incomplete.

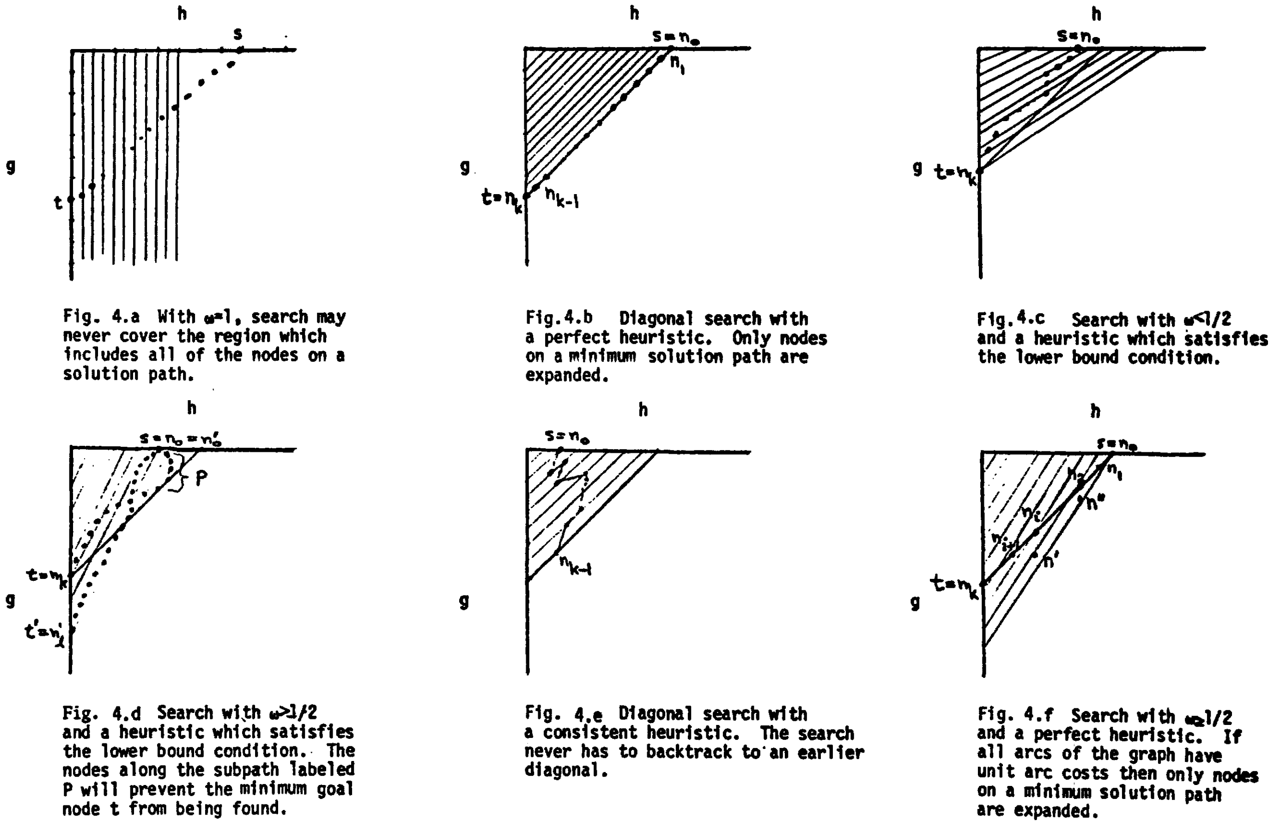


Fig. 4.a With $\omega=1$, search may never cover the region which includes all of the nodes on a solution path.

Fig. 4.b Diagonal search with a perfect heuristic. Only nodes on a minimum solution path are expanded.

Fig. 4.c Search with $\omega<1/2$ and a heuristic which satisfies the lower bound condition.

Fig. 4.d Search with $\omega>1/2$ and a heuristic which satisfies the lower bound condition. The nodes along the subpath labeled P will prevent the minimum goal node t from being found.

Fig. 4.e Diagonal search with a consistent heuristic. The search never has to backtrack to an earlier diagonal.

Fig. 4.f Search with $\omega\geq 1/2$ and a perfect heuristic. If all arcs of the graph have unit arc costs then only nodes on a minimum solution path are expanded.

Figure 4—Illustrations of different search strategies with various conditions on the heuristic. Although not explicitly shown in the figures, all modes which lie in a region of search and have ancestors forming a path back to the start node which also lie in the region of search are expanded

The admissibility theorem tells us the conditions under which the first solution found is a minimum cost solution.

THEOREM 2. [Admissibility] Let h satisfy the lower bound condition, $h \leq h_p$, where h_p is the perfect heuristic. Then search with h is admissible for δ-graphs iff, $\omega \epsilon [0,\frac{1}{2}]$.

PROOF: Let us begin by thinking in terms of the nodes positioned in the plane as defined by the perfect heuristic h_p. Since $g(n_i)+h_p(n_i)$ equals the minimum solution cost for each node n_i which is on a minimum solution path $s=n_0, \ldots, n_k=t$, all such n_i must lie on the minimum cost diagonal of the plane (see Figure 4(b)). Since $g(n')+h_p(n')$ is greater than the minimum solution cost for each node n' which does not lie on a minimum solution path, all such n' lie outside the triangular region defined by the minimum cost diagonal. Thus, no nodes lie within this triangular region. If the heuristic is not perfect, but does satisfy the lower bound condition, then the nodes $n_0, \ldots, n_k$ are all pulled to the left and lie in the triangular region (some may lie on the boundary). Thus, diagonal, upper diagonal, and any search with $\omega \geq \frac{1}{2}$ (see Figure 4(c)) will have expanded $n_0, \ldots, n_{k-1}$ by the time search reaches the location of the minimum goal node t, and therefore t will be found before any other goal node. Hence if $\omega \epsilon [0,\frac{1}{2}]$ the search is admissible.

However, if $\omega > \frac{1}{2}$ (see Figure 4(d)) the region covered by the search can include all of the nodes on a nonminimum path like $s-n_0', \ldots, n_l'=t'$ (and thus find a nonminimum solution), before it covers all of the nodes on the minimum path $s=n_0, \ldots, n_k$. In Figure 4(a) the nodes along the subpath (of the minimum solution path) labeled P lie outside the region which includes all of $s=n_0', \ldots, n_l'=t'$. These nodes (and all of the successors along the minimum path) will not be expanded, and the minimum goal node will not be found. Thus in $\omega \epsilon [0,\frac{1}{2}]$ the search is not admissible.

The optimality theorem tells us that for all of the values of ω which give admissible searches, the use of better heuristics will result in improved searches.

THEOREM 3. [Optimality] Let $\omega \epsilon [0,\frac{1}{2}]$, and $h_2 < h_1 \leq h_p$. Then search with h_2 expands every node expanded by search with h_1 for all δ-graphs that contain a minimum solution.

PROOF: Again let us begin with the nodes positioned in the plane for the perfect heuristic, where nodes on a minimum solution path lie on the minimum cost diagonal and all others lie outside the triangular region. Now think in terms of the movement of the nodes in the plane which results from using the heuristics h_1 and h_2 instead of h_p. Since $h_2 < h_1$, h_2 pulls all of the nodes into the triangular region which are pulled there by h_1, and others besides. Not all of the nodes which lie inside the triangular region will be expanded. Only those for which there exists a path back to a start node such that all of the nodes on this path also lie in the triangular region will be expanded. However, all of the nodes pulled in by h_1 that are expanded by h_1 will also be expanded by h_2, because those paths back to a start node whose nodes are placed in the triangular region by h_1 will also be placed there by h_2. Thus search with h_2 expands all of the nodes expanded by search with h_1, and diagonal search (i.e., $\omega=\frac{1}{2}$) is optimal.

For $\omega < \frac{1}{2}$ the reason for optimality is similar. The poorer heuristic pulls more nodes inside the triangular region than the better heuristic, only now the triangular region in question is as shown in Figure 4(c).

The completeness and admissibility theorems were stated as iff conditions in terms of the subinterval of [0,1] in which the properties held. This is not done for the optimality theorem because optimality is defined for admissible heuristics, and admissibility holds only in the subinterval $[0,\frac{1}{2}]$. Thus since $\omega \epsilon [\frac{1}{2},1]$ eliminates admissibility, optimality in this subinterval is not a queston.

If the hypothesis of the optimality theorem is changed from $h_2 < h_1 \leq h$ to $h_2 \leq h_1 \leq h$, then it is possible that search with h_2 may not expand a node n which lies on the minimum cost diagonal which is expanded by search with h_1. This is because a search strategy resolves ties arbitrarily, and search with the poorer heuristic h_2 may choose to expand a node n which is tied for merit with a minimal goal node, while search with the better heuristic h_1 chooses not to expand n. Thus, if $h_2 \leq h_1 \leq h$ then search with h_2 expands all the nodes expanded by search with h_1 except possibly for a set of nodes which have the same merit as a minimum goal node.

Let us now turn to the consistency property of a heuristic function. It has been pointed out that in general a search strategy does not expand the nodes of the graph according to the merit ordering. Thus, in general, diagonal search does not expand a node as soon as it falls within the region which has been searched. It must continually backtrack to earlier diagonals to expand nodes. Consistency of the heuristic is a sufficient condition to prevent this from happening.

THEOREM 4. If h is consistent then diagonal search (i.e., $\omega=\frac{1}{2}$) never has to backtrack to an earlier diagonal to expand a node.

PROOF: If h is consistent, then if $k(n,n')$ exists then $h(n)-h(n') \leq k(n,n')$. What does this condition mean in terms of the coordinate representation of the search space?

$$\begin{aligned} & h(n)-h(n') \leq g(n')-g(n) \\ \Leftrightarrow\ & g(n)+h(n) \leq g(n')+h(n') \\ \Leftrightarrow\ & h(n)-h(n') \leq k(n,n') \\ \Leftrightarrow\ & f(n) \leq f(n') \end{aligned}$$

Thus consistency means that if n precedes n' in the graph (which is a necessary and sufficient condition for $k(n,n')$ to exist), then $f(n) \leq f(n')$. In terms of the coordinate representation this means that n appears on the same or an earlier diagonal. Hence diagonal search moves from diagonal to diagonal to expand nodes, and never has to backtrack to an earlier diagonal to expand a node (see Figure 4(e)).

An important consequence of Theorem 4 is that if a heuristic is consistent then an ordered search strategy will never have to expand a node more than once; the first path found to any node is the minimum cost path to that node. Thus, when the heuristic is consistent, Case 3 (in the algorithm given in Figure 1), where $m \epsilon S$ and the new merit is better than the old merit, will not arise.

Sometimes the concept of a monotonic evaluation function is used instead of consistency. An evaluation function f is monotonic if for all n and n', $n \leq n' \Rightarrow f(n) \leq f(n')$, and $h(n') = 0$ when n' is a goal node; where $n < n'$ means that n precedes n' in the graph, and $n \leq n'$ means that either n precedes n' or n equals n'. For $\omega = \frac{1}{2}$ the concept of monotonicity is equivalent to that of consistency.

At one time it was thought that the optimality property required the consistency assumption. Recall that the proof of the optimality theorem depended on the poorer heuristic h_2 pulling all of the nodes into the triangular region which are pulled in by the better heuristic h_1. Thus all of the nodes pulled in and expanded by h_1 were pulled in and expanded by h_2. If the consistency of h_1 is assumed, then every node pulled in by h_1 is also expanded by h_1. Thus we can say every node pulled in by h_1 is pulled in and expanded by h_2. However, it is clear that the optimality property does not depend on h_1 expanding every node which it pulls into the triangular region. That is, the consistency assumption is not necessary for optimality.

THEOREM 5. [Perfect Heuristic] Suppose that all arcs of the graph have unit costs. Then search with $f_\omega = (1-\omega)g + \omega h$ only expands nodes on a minimum solution path iff $[\frac{1}{2}, 1]$.

PROOF: As indicated earlier, a perfect heuristic places no nodes inside the triangular region defined by the minimum cost diagonal, and places all nodes on all minimum solution paths on the minimum cost diagonal itself. No nodes will be expanded until the area covered by the search includes the start node (see Figure 4(f)). To prove the theorem we must show that at each stage of the search the node that is expanded lies on the minimum cost diagonal.

Let $s = n_0, n_1, \ldots, n_k = t$ be a minimum solution path. Clearly the first stage expands a node on the minimum cost diagonal since the node expanded is n_0. Suppose that $n_0, n_1, \ldots, n_i$ are the only nodes which have been expanded at the ith stage. Then since all arcs have unit arc costs the successor of n_i which is on the minimum solution path namely n_{i+1}, lies on the diagonal and a successor n' which is not on the minimum solution path lies off of the diagonal to the right of n_{i+1} (see Figure 4(f)). Thus n_{i+1} has better merit than the other successors of n_i. n_{i+1} has better merit than any previously unexpanded successor n'', because n_i has better merit than n'', and n_{i+1} has better merit than n_i. Hence, n_{i+1} will be expanded at stage i+1, and only nodes on the minimum solutions path are expanded.

If $\omega \epsilon [\frac{1}{2}, 1]$ then it is possible for the previously unexpanded successor n'' to have better merit than n_{i+1}. Hence a node not on the minimum solution path may be expanded.

If the unit arc costs assumption is removed from the hypothesis of Theorem 5 the conclusion is not true. This is because the node n' may be in the region which gives it a better merit than n_{i+1}, and thus a node off of the minimum solution path will be expanded. It is clear that the unit arc cost assumption can be replaced by a constant arc cost assumption. Figure 5 illustrates that Theorem 5 does not generalize to arbitrary arc costs.

All values of $0 \leq \omega \leq \frac{1}{2}$ are equally good in the sense that completeness, admissibility, and optimality all hold for $\omega \epsilon [0, \frac{1}{2}]$ provided h satisfies the lowerbound condition. However, it is clear from an examination of the regions expanded by searching with various values of $\omega \epsilon [0, \frac{1}{2}]$ that $\omega = \frac{1}{2}$ is best, because it expands fewest nodes.

Ignoring admissibility and optimality for the mo-

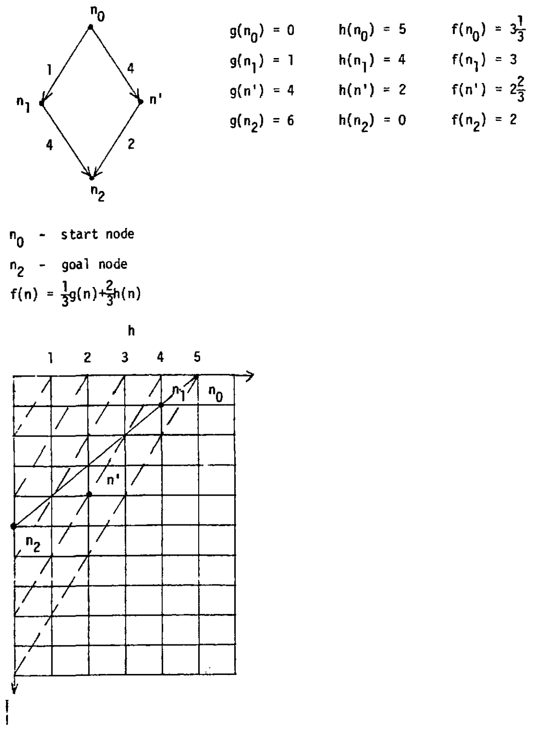


Figure 5—An illustration that Theorem 5 does not generalize to arbitrary arc costs. Searching the above graph with the perfect heuristic using $\omega = 2/3$ not only expand the node n' which is off of the minimum solution path, it even finds the nonminimal solution n_0, n', n_2

ment, there seem to be intuitive reasons for choosing $\omega=1$, or at least for choosing $\omega>\frac{1}{2}$. Recall that g is the cost-to-date component of the evaluation function, and that h estimates the cost to the nearest goal. An intuitive reason for choosing $\omega=1$ is as follows. Since the search has progressed to the point of generating n, why should we concern ourselves with the cost that has been incurred to get there? Perhaps we should only concern ourselves with what is the expected cost to get to a goal.

On the other hand, in addition to the fact that we may desire a minimum solution, which is not guaranteed when $\omega>\frac{1}{2}$, there is the following reason for including a g-component in the evaluation function. Sometimes it is difficult to construct a good heuristic function for a problem, and in these cases the presence of a g-component does not hinder the search, and sometimes helps it. Consider, for example, a heuristic function h which is of bounded error ($h_p(n)-\epsilon\leq h(n)\leq h_p(n)+\epsilon$, for some $\epsilon>0$ and all n), and which is defined in a way that deliberately misleads the search by being as optimistic as possible for nodes off the minimum solution path and as pessimistic as possible for nodes on the minimum solution path. It has been shown that for such a heuristic, diagonal search ($\omega=\frac{1}{2}$) expands fewer nodes in obtaining a solution than does pure heuristic search ($\omega=1$). This is an extreme example, but it does illustrate how a g-component can act as a stabilizing source when a poor heuristic is used.

BIBLIOGRAPHICAL REMARKS AND CONCLUSION

Graph representations have long been recognized as important models in problem solving. Discussion of both early and more recent use of graph representations can be found in the books by Ernst and Newell,[4] Barnerji,[5] Nilsson,[6] and Slagle.[7] These books along with the articles by Amarel,[8] Michie,[9] and Sandewall[10] also contain general discussions of ordered search strategies.

The admissibility and optimality of ordered search strategies and the concepts related to these theorems are due to Hart, Nilsson, and Raphael.[1,11] Algebraic proofs of these theorems also appear in Nilsson.[6] In this reference a δ-graph is defined to be a graph whose arc costs are bounded away from zero. The definition used in this paper which restricts the partial sums of the arc costs along an infinite path from being unbounded is more general, since it allows finitely many arcs of zero cost. This definition is based on the concept of a δ-finite merit ordering which was used by Kowalski,[3] who also was the first to use the geometric representation. Pohl[12] investigated the evaluation function $f_\omega=(1-\omega)g+\omega h$ in some experiments on the 15-puzzle. Doran and Michie[13] also experimented with the 15-puzzle. The theorem on completeness was proved by Pohl.[2] The theorem on the perfect heuristic and the argument for including a g-component in evaluation function are also due to Pohl.[12] To the best of our knowledge, the fact that the theorem on the perfect heuristic does not generalize to arbitrary arc costs has not previously been pointed out.

Hart, Nilsson, and Raphael,[1,11] developed ordered search strategies for directed graphs, which are used to represent state-space representations. Chang and Slagle[14] developed ordered search strategies for AND/OR graphs, which are used for problem-reduction representations. Ordered search strategies for theorem-proving graphs, which are used to represent state-space representations with multiple-input operators, were developed by Kowalski,[3] and also by Michie and Siebert.[15] An illustration of how the direction in which a problem space is searched determines whether a single problem representation is viewed either as a state-space or a problem-reduction representation appears in VanderBrug and Minker.[16]

The formal properties of ordered search algorithms, which we proved using the geometric representation, are of limited interest to the developers of a practical problem solving system. Completeness is rarely a big consideration in the design of such a system. Usually what keeps a search from being successful is exhaustion of the available resources, not the incompleteness of the search strategy. Admissibility is only sometimes an important consideration, since often a solution which is approximately minimum is sufficient. Thus, one can use a heuristic which only approximately satisfies the lowerbound condition (i.e., $h\leq h_p+\epsilon$), and expect to get a solution which is approximately minimum. Such a search is less conservative (in the sense that it takes more chances) than an admissible one, and frequently will find an approximately minimum solution before the admissible search finds a minimum one.

Nonetheless, the theorems are important formal properties, and help to unify the work done in this area. We believe that the geometric approach to the presentation of these formal properties is an intuitive alternative to the algebraic approach.

ACKNOWLEDGMENTS

The support of the National Aeronautics and Space Administration under Grant NGR-21-002-270, and the assistance of L. Kanal, J. Minker and A. Rosenfeld is gratefully acknowledged.

REFERENCES

1. Hart, P. N., N. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans, Sys. Sci. Cybernetics*, Vol. S.S.C.-4, No. 2, July 1968, pp. 100-107.
2. Pohl, I., "Heuristic Search Viewed as Path Finding in a

Graph," *Artificial Intelligence Journal*, Vol. 1, No. 3, Fall 1970, pp. 193-204.

3. Kowalski, R., "Search Strategies for Theorem-Proving," In: Meltzer, B. & Michie, D. (eds.), *Machine Intelligence 5*, American Elsevier, New York, 1970, pp. 181-201.
4. Ernst, G. W. and A. Newell, *GPS—A Case Study in Generality and Problem Solving*, Academic Press, New York, 1969.
5. Banerji, R. B., *Theory of Problem Solving*, American Elsevier, New York, 1969.
6. Nilsson, N. J., *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, 1971.
7. Slagle, J. R., *Artificial Intelligence: The Heuristic Programming Approach*, McGraw-Hill, New York, 1971.
8. Amarel, S., "An Approach to Heuristic Problem-Solving and Theorem Proving in the Propositional Calculus," In: J. Hart and S. Takasu (eds.), *Systems and Computer Science*, U. of Toronto Press, Toronto, 1967.
9. Michie, D., "Heuristic Search," *Computer Journal*, Vol. 14, No. 1, pp. 96-102.
10. Sandewall, E., "Heuristic Search: Concepts and Methods," In: Findler, N. V. & Meltzer, B. (eds.), *Artificial Intelligence and Heuristic Programming*, American Elsevier, New York, 1971, pp. 81-100.
11. Hart, P. N., N. Nilsson and B. Raphael, "Correction to 'A Formal Basis for the Heuristic Determination of Minimum Cost Paths'," *SIGART Newsletter*, Dec. 1972.
12. Pohl, I., "First Results on the Effect of Error in Heuristic Search," In: Meltzer, B. & Michie, D. (eds.), *Machine Intelligence 5*, American Elsevier, New York, 1970, pp. 219-236.
13. Doran, J. E. and D. Michie, "Experiments with the Graph Traverser Program," *Proc. R. Soc. (A)*, Vol. 294, pp. 235-259.
14. Chang, C. L. and J. R. Slagle, "An Admissible and Optimal Algorithm for Searching AND/OR Graphs," *Artificial Intelligence Journal*, Vol. 2, No. 2, Fall 1971, pp. 117-128.
15. Michie, D. and E. E. Siebert, "Some Binary Derivation Systems," *JACM*, Vol. 21, No. 2, April 1974, pp. 175-190.
16. VanderBrug, G. J. and J. Minker, "State-Space, Problem-Reduction, and Theorem Proving—Some Relationships," *CACM*, Vol. 18, No. 2, February 1975, pp. 107-115.

Another algorithm for reducing bandwidth and profile of a sparse matrix

by W. F. SMYTH
International Labor Office
Geneva, Switzerland

and

ILONA ARANY
Computing Center of the Ministry of Labor
Budapest, Hungary

ABSTRACT

The paper describes a new bandwidth reduction method for sparse matrices which promises to be both fast and effective in comparison with known methods. The algorithm operates on the undirected graph corresponding to the incidence matrix induced by the original sparse matrix, and separates into three distinct phases: (1) determination of a spanning tree of maximum length, (2) modification of the spanning tree into a free level structure of small width, (3) level-by-level numbering of the level structure. The final numbering produced corresponds to a renumbering of the rows and columns of a sparse matrix so as to concentrate non-zero elements of the matrix in a band about the main diagonal.

INTRODUCTION

As electronic computers make possible computations on ever larger data sets, it has come to be realized [1,p17,2] that most large matrices are, in the nature of things, sparse; more precisely, that a matrix of (large) order n will generally contain only Kn non-zero elements, where K tends to decrease as n increases (K is often as little as 2 or 3 and only rarely greater than 20). There has been, accordingly, during the last ten years, a good deal of research into techniques for efficient computer handling of large sparse matrices. These techniques may be separated into two classes:

T_1: techniques which deal with the given sparse matrix, more or less directly, in a sparse form (some typical approaches are surveyed in [2] and [3]);

T_2: techniques which transform the given sparse matrix into a band form (which may then be processed further by efficient and well-known band matrix algorithms).

Initially "direct" techniques T_1 attracted more interest, and the utility of "band" techniques T_2 was occasionally questioned.[4] Research into T_2 techniques was spurred however by the work of Cuthill-McKee,[5] who described an effective bandwidth reduction algorithm with execution time linear in Kn. Instead of dealing directly with the given matrix, Cuthill-McKee (hereafter called CM) dealt with the numbered graph whose connections correspond to the given matrix's zero/non-zero structure; they renumbered the vertices of this graph, and this renumbering therefore defined the interchanges of rows and columns required to transform the original sparse matrix into a band matrix.

The CM method was later modified in various ways, especially by Rose,[6] who also contributed an important analysis of the application of Gaussian elimination to band matrices, showing in particular the importance of the profile.

The algorithm presented here, called the SA algorithm, is of class T_2. It results from efforts to improve on previous work,[7] and has been directly stimulated by the approach suggested by Smyth-Benzi (hereafter called SB)[8] as well as by the related algorithm published independently by Gibbs-Poole-Stockmeyer (hereafter called GPS).[9] Like the CM and GPS algorithms, SA renumbers the vertices of a given numbered graph with the objective of minimizing the maximum difference between numbers assigned to connected vertices. Also in common with these algorithms, SA reduces profile as well as bandwidth[6] and may be applied to non-symmetric as well as to symmetric matrices.

TERMINOLOGY

We use the term *graph*, and the symbol G or G(V, E), to denote a finite connected * undirected graph without

* The algorithm may easily be modified to deal separately with the disjoint components of a single graph.

loops or multiple edges defined on a *vertex* set V of cardinality $n=|V|>1$ and an *edge* set E of cardinality $m=|E|$. For any distinct pair of vertices $u, v \epsilon V$, we define the usual *distance* function $\rho(u,v)$ to be the number of edges on the shortest *path* from u to v. For a single vertex u, we adopt the convention $\rho(u,u)=0$; for unconnected vertices u,v, we set $\rho(u,v)=\infty$; if $\rho(u,v)=1$, then u and v are said to be *adjacent*. The *diameter* of G is defined by

$$\text{diam}(G)=\max_{u,v\epsilon V}[\rho(u,v)].$$

Since we assume G is connected, $\text{diam}(G)\leq n\text{-}1$.

Apart from such basic terms, we need for our purposes here to define four main concepts: level structure, free level structure, numbering, and bandwidth.

A *level structure* LS of a graph G(V,E) is an assignment of the vertices of V into sets, called *levels* $L_1, L_2, \ldots, L_\lambda$, such that

(1) L_1 contains at least one vertex;
(2) for each $k=2,3,\ldots,\lambda$, L_k contains every vertex not in a previous level which is adjacent to some vertex of L_{k-1}.

It follows from this definition that if G is connected, LS contains all n vertices of V; that the levels are disjoint; that $1\leq\lambda\leq\text{diam}(G)+1$; and that LS is determined uniquely by the choice of vertices in L_1. We may therefore unambiguously denote LS by $LS(L_1)$ or, when $L_1=\{u\}$, by LS(u). We observe in fact that level L_k of $LS(L_1)$ consists of exactly those vertices which occur in the kth level of every *spanning forest* (SF) grown from the vertex set L_1 (for a description of this process see Reference 5). $LS(L_1)$ therefore corresponds (one-many) to the $SF(L_1)$, and LS(u) to the *spanning trees* ST(u). Whenever edges are not important, then, we may refer to LS and SF/ST interchangeably.*

A *free level structure* FLS of a graph G(V,E) is an arrangement of all n vertices of V into λ levels $L_1, L_2, \ldots, L_\lambda$, such that

(1) no level is empty;
(2) if $u \epsilon L_k$ then all vertices adjacent to u are in either L_{k-1}, L_k, or L_{k+1}.

We note that in this case also $1\leq\lambda\leq\text{diam}(G)+1$, but that FLS(u) is no longer uniquely determined (Figure 1). For either LS or FLS we speak of the *width of level k*, $w(L_k)=|L_k|$, and the *width of the structure*, w(LS) or $w(FLS)=\max_{1\leq k\leq\lambda}w(L_k)$. λ is called the *length* of the structure.

Following GPS, we now define a *numbering* $\alpha=\alpha(V)$ of G(V,E) to be a one-one map of V onto the first n natural numbers $\{1,2,\ldots,n\}$. For a given numbering α, we may define $\delta_\alpha(G)$, *the bandwidth of the graph G relative to the numbering* α, to be

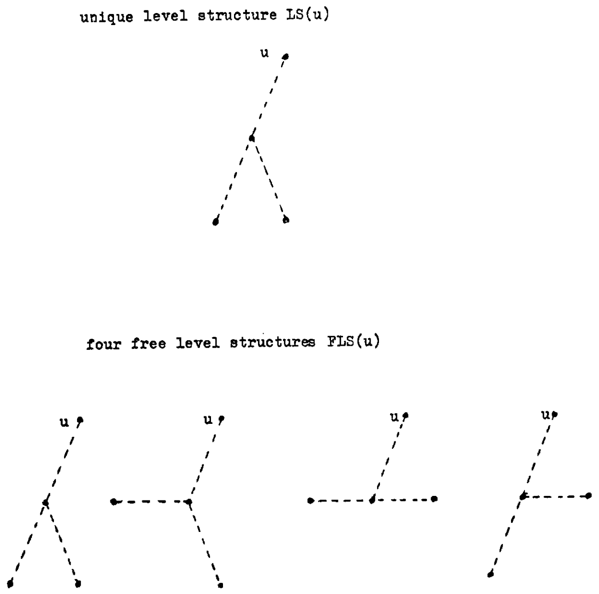

unique level structure LS(u)
u
four free level structures FLS(u)
u
u
u
u

Figure 1

$$\delta_\alpha=\delta_\alpha(G)=\max_{(u,v)\epsilon E}|\alpha(u)-\alpha(v)|.$$

The *bandwidth of G* is then

$$\delta=\delta(G)=\min_{\text{all}\,\alpha}[\delta_\alpha(G)].$$

GENERAL DESCRIPTION OF THE SA ALGORITHM

Our goal is therefore, given G(V,E), to find a numbering $\alpha=\alpha(V)$ such that $\delta_\alpha=\delta(G)$. Our (heuristic) algorithm approaches this problem in three distinct phases:

I *Find LS*
Using the SB method,[8] a level structure LS $=LS(u)$ is found such that $\lambda=\text{diam}(G)+1$.

II *Find FLS*
Using a subset M ($|M|=\min\{|L_\lambda|, |\overline{n}/\overline{\lambda}|\}$) of the vertices of level L_λ in LS(u), a new level structure, denoted $LS'=LS'(M)$, is grown. LS′ is also of length λ. LS and LS′ are systematically compared, and an FLS is determined such that $w=w(FLS)$ is small (if possible, $w=|\overline{n}/\overline{\lambda}|$).

III *Number FLS*
FLS is numbered on a level-by-level basis; that is, first, the integers $\{1,2,\ldots,w_1\}$ are assigned to the vertices of the first level of FLS (w_k denotes the width of the kth level of FLS);

* We dwell on this point to avoid confusion. The definition of LS given here is more restrictive than the original definition given in Reference 7, but is compatible with the SF/ST usage of Reference 8. The GPS definition of LS Reference 9 is compatible with Reference 7, and what GPS call the "level structure rooted at u" is identical to our LS(u).

second, the integers $\{w_1+1, w_1+2, \ldots, w_1+w_2\}$ are assigned to the vertices of the second level; and so on, until all vertices have been numbered. The algorithm makes use of knowledge of the edges joining vertices of successive levels in an effort to minimize bandwidth: it searches for a level-by-level numbering α such that the corresponding bandwidth satisfies

$$\delta_\alpha \leq w+\Delta,$$

where Δ successively takes the values $0,1,\ldots, w-1$. The first numbering found which satisfies this condition is the required numbering α.

The justification of algorithms such as this is partly theoretical, partly experimental, and numerous variations in strategy are possible. In practical terms, we are trying to find a "reasonably good" numbering without needing to investigate all of the n! different possible numberings; our strategy therefore is always influenced by estimates, often very rough in nature, of the additional benefit to be expected from the additional effort expended. We will find an example of this kind of strategic thinking in Phase II of SA: we do not carry out an exhaustive search to find the FLS of truly minimum width, even though such a search might not on the average be too laborious, simply because (1) it seems that an exhaustive search would not be likely to decrease w; (2) the result might be merely to decrease w by 1, but not decrease δ_α, and in addition make numbering more lengthy and difficult. On the other hand, in Phase I, we propose using the SB algorithm [8] instead of the GPS pseudo-diameter algorithm,[9] because the former guarantees finding a longest spanning tree at (apparently) no additional cost. On the theoretical side, the basic justification for level-by-level numbering is the result of Arany-Szóda [10] that corresponding to every numbering α of G there exists at least one FLS whose width $w=\delta_\alpha$, and which may be numbered on a level-by-level basis to yield the numbering α. Since the case $\delta_\alpha=\delta$ is included in this result, it follows that level-by-level numbering of an FLS is an acceptable approach to bandwidth reduction, in that it does not exclude any minimum case. The result however does not point to any particular FLS-growing or FLS-numbering algorithms.

PHASE I OF SA: FIND LS

As noted above, the SB diameter algorithm is proposed here, because it guarantees finding $\lambda=\text{diam}(G)+1$ and apparently is comparable to the GPS pseudo-diameter algorithm in execution time. The SB algorithm is described fully in Reference 8 and is not included here*.

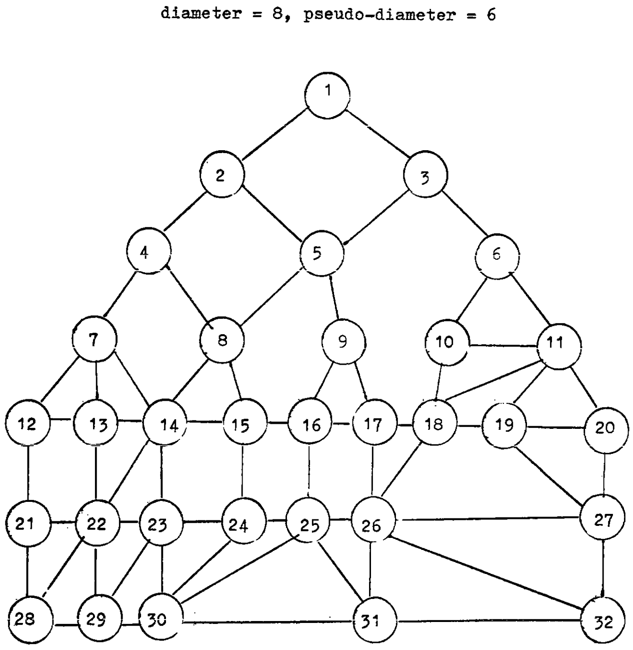


Figure 2

GPS state in Reference 9 that their pseudo-diameter algorithm found the true diameter in all test cases. Figure 2 illustrates a graph of diameter 8 whose GPS pseudo-diameter is 6. The GPS algorithm would require the growth of 6 spanning trees to achieve this result, the SB algorithm 3 spanning trees (starting vertex ①). Removal of edges (4,8) and (5,9) from Figure 2 and insertion of edge (4,5) would permit GPS to find a pseudo-diameter 7 at a cost of growing 4 spanning trees. For a symmetrized version of the Curtis matrix[4] (Figure 3), SB finds diameter 7 at a cost of 6 spanning trees (starting vertex ①), and GPS finds pseudo-diameter 7 from each of six starting vertices of minimum degree at an average cost of 5.5 spanning trees. See Table I.

A more detailed analysis of the application of the GPS pseudo-diameter algorithm to the Curtis matrix

TABLE I—Comparison of SB and GPS

	(pseudo-) diameter		number of LS grown	
example	SB	GPS	SB	GPS
Figure 2	8	6	3	6
Figure 2 (mod.)	8	7	3	4
Curtis matrix	7	7	6	5.5

* We do however provide a correction to the algorithm SPAN (h,i) given in the Appendix of Reference 8. In step 7 replace the ending "." with "gotoll.". Step 8 should read "[Is the vertex connected to jorig in level g+1?] *If h=1* and level (h, elist(j)) =g+1, goto10; otherwise gotoll.". Interchange steps 9 and 10.

FLS corresponding to Curtis matrix

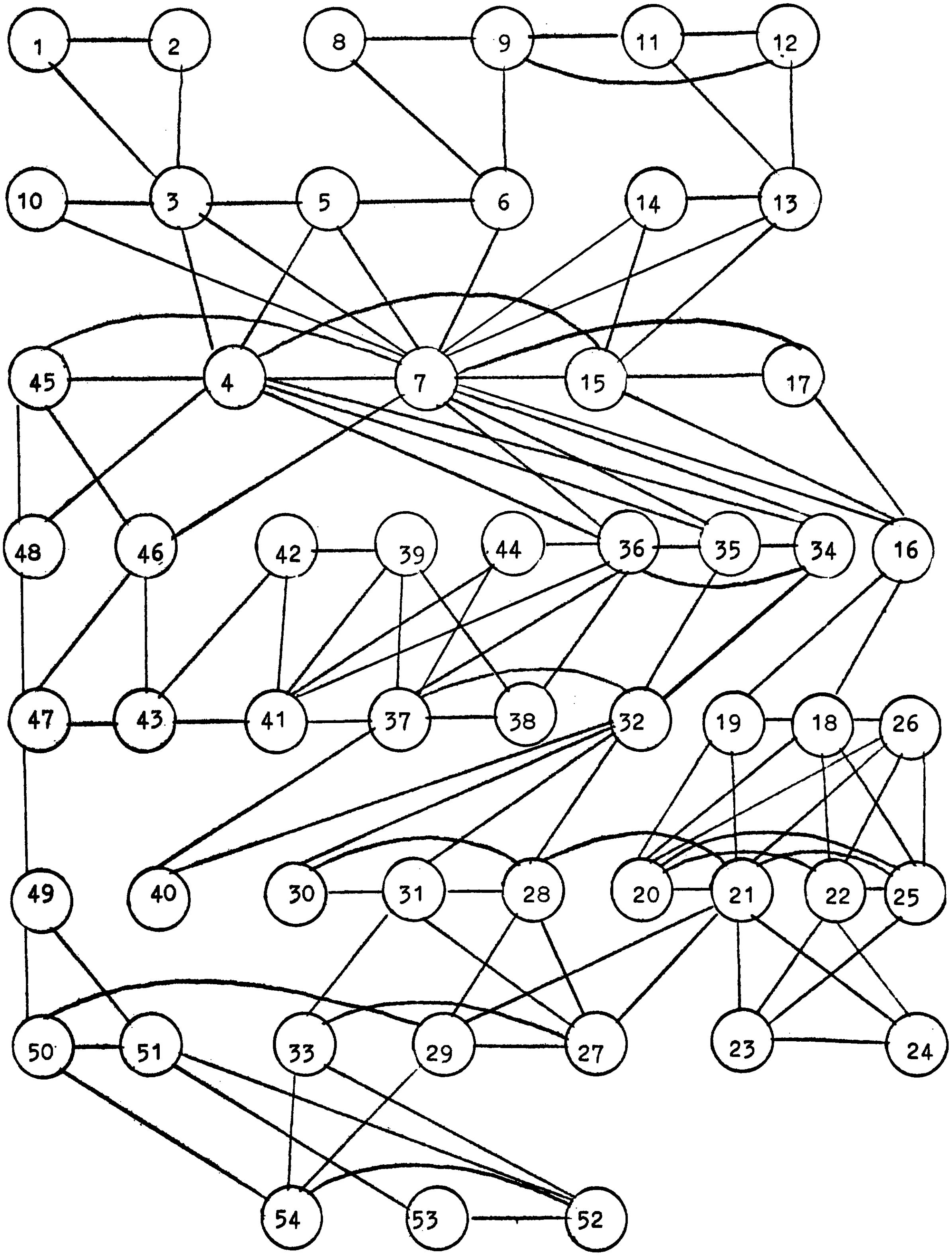


Figure 3

shows that 48 of the 54 vertices would, used as starting root vertices, yield maximum pseudo-diameter 7; of these 48 vertices, the 9 extreme vertices yield $\lambda=8$ immediately, 22 yield $\lambda=7$ but contain an extreme vertex in L_7, and 15 yield $\lambda=6$ with an extreme vertex in L_6 (there are two other special cases). These statistics appear to be related to the success of the GPS algorithm, and it would be interesting to know if for much larger graphs also ($n=500$ to 5000) such a high percentage of optimal starting root vertices occurs.

GPS initially searches V for a vertex of lowest degree as starting root vertex, and from the corresponding L_λ selects additional root vertices in increasing order of degree. SB includes no such searches.

PHASE II OF SA: FIND FLS

The algorithm described below is an attempt to deal efficiently with difficult cases which may arise when G is not "well" connected (Figure 4); that is, when a number of vertices swing into different levels in the level structure LS′ grown from vertices of L_λ in LS(u). It seems that the growing of this "reverse" level structure (step (1) of the following algorithm) will normally accomplish what GPS accomplish in phase II of their algorithm; the other steps in the algorithm given below make use of the same information proposed by GPS, but in a more flexible manner.

Given LS(u) of length $\lambda=\mathrm{diam}(G)+1$, phase II proceeds as follows:

(1) [Grow "reverse" LS′.] For $R=|\bar{n}/\bar{\lambda}|$,* choose a set of vertices $M<L_\lambda$ such that $|M|=\min\ [R, w(L_\lambda)]$ and grow LS′(M) with levels L_k' of width $w'(k)$, $k=1,2,\ldots,\lambda$ ($L_1'=M$).

(2) [Calculate initial value of T.]
$T\leftarrow\Sigma_{1\leqq k\leqq\lambda}\min\ [R-w'(k),0]$; if $T=0$, goto (16).
[Two criteria are used to determine whether the movement of a vertex of FLS(M) from one level to another yields a "better" FLS: the value T given here and the value $T'=\Sigma[R-w'(k)]^2$. The FLS is "better" if the vertex movement increases T (by unity) or decreases T′. T is also an "absolute" criterion, since any FLS for which $T=0$ is accepted; T′ is not calculated explicitly because it is not "absolute", so that only $\Delta T'$ is required (see step (12)). Both T and T′ may be used since T′ decreases iff T does not increase.]

(3) [Calculate level pairs.] As in Reference 9, associate with each vertex $v\in V$ a *level pair* (g_v,h_v), where g_v is the level of v in LS′(M), where $h_v=\lambda-k_v+1$, and k_v is the level of v in LS(u).

(4) [Separate "movable" vertices into r connected components.] As in Reference 9, separate all vertices such that $g_v=h_v$ into r disjoint connected components $C_1,C_2,\ldots,C_r$ of cardinality $c_1,c_2,\ldots,c_r$, respectively, arranged so that $c_1\leq c_2\leq\ldots\leq c_r$.

(5) If $r=0$, goto (16).

(6) [Organize storage for vertices in component C_r.] Associate with each vertex $x_{rj}\epsilon C_r$, $j=1,2,\ldots,c_r$, an *identifier* $[v(j),g(j),h(j)]$, where $v(j)$ is a pointer to the level pair of the vertex $v=x_{rj}$, and $[g(j),h(j)]$ is the level pair of x_{rj} (that is, $g(j)=g_{v(j)}$, $h(j)=h_{v(j)}$); order the vertices so that $g(1)\geq g(2)\geq\ldots\geq g(c_r)$.

(7) [Initialize vertex movement parameters for C_r.] $j\leftarrow 1$; $\Delta T\leftarrow 0$, $\Delta T'\leftarrow 0$; for each $k=1,2,\ldots,\lambda$, $w(k)\leftarrow w'(k)$.
[For each C_r, we need to provide temporary storage for the level widths $w'(k)$: this temporary storage into $w(k)$ is used to keep track of moves made since the last acceptance of a vertex movement (last increase of T or decrease of T′).]

(8) [Move vertex x_{rj} from level $g(j)$ to $g(j)-1$.] Perform MOVE-VERTEX(j,γ) as follows:
(8.1) $\gamma\leftarrow g(j)\leftarrow g(j)-1$;
(8.2) $w(\gamma)\leftarrow w(\gamma)+1$, $w(\gamma+1)\leftarrow w(\gamma+1)-1$;
(8.3) let j' successively take the values $j+1,\ldots,c_r,1,\ldots,j$; for the first value j' such that $g(j')>h(j')$, set $j\leftarrow j'$;
(8.4) if there is no such j', set $j\leftarrow 0$.
[Given j, MOVE-VERTEX(j,γ) returns the new level γ of the moved vertex x_{rj}, together with the next admissible value of j. The vertex movements take place in a "cascade" from higher-level vertices to lower-level vertices, in accordance with the observation that under these circumstances a vertex x_{rj} may always be moved to the preceding level, provided that $g(j)>h(j)$ (initially, of course, $g(j)>h(j)$ for every $j=1,2,\ldots,c_r$). Note that not all possible configurations of C_r are necessarily covered by the movements included here.]

(9) [Calculate ΔT.] $\Delta T\leftarrow\Delta T+$ (if $w(\gamma+1)>R-1$, 1; otherwise, 0) $-$ (if $w(\gamma)>R$, 1; otherwise 0).
[The full expression for ΔT is
$$T=-\min\{R-[w(\gamma)-1],0\}+\min\{R-w(\gamma),0\}-\min\{R-[w(\gamma+1)+1],0\}+\min\{R-w(\gamma+1),0\}.]$$

(10) [For increased T, store present arrangement of vertices into levels.] If $\Delta T=1$, $T\leftarrow T+1$, $\Delta T\leftarrow 0$, and perform STORE-PATTERN; otherwise, goto (12).
[STORE-PATTERN does the following: for $j=1,2,\ldots,c_r$, $g_{v(j)}\leftarrow g(j)$; for $k=1,2,\ldots,\lambda$, $w'(k)\leftarrow w(k)$; $\Delta T'\leftarrow 0$. Note that $\Delta T'$ is reset after every increase of T, but that acceptance based on $\Delta T'$ does not reset ΔT (step (13)).]

(11) If $T=0$, goto (16).

(12) [Calculate $\Delta T'$.] $\Delta T'\leftarrow\Delta T'+[w(\gamma)-w(\gamma+1)-1]$.

(13) If $\Delta T'<0$, perform STORE-PATTERN.

* SB propose rather $|\bar{n}/\bar{\lambda}|$.

(14) [More vertex movements possible?] If $j>0$, goto (8).

(15) [More components of "movable" vertices?] $r \leftarrow r-1$, goto (5).

(16) Exit.

[The FLS(M) which best satisfies the T and T' criteria has now been determined. Each vertex v of FLS(M) is placed into the level specified by g_v, and the width of each level L_k' is given by $w'(k)$.]

Figure 4 illustrates the result of the application of the algorithm to an LS with many movable vertices. In this case only three vertices would need to be moved before FLS became acceptable ($T=0$). In the example of Figure 2, an FLS of width 5 would be accepted ($T=-3$) after movement of two vertices. For the two dimensional grid discussed in detail by GPS, phase II yields an optimal FLS=LS(M) after step (1), and no vertex movements are required. For the Curtis matrix, on the other hand, consideration of the connected components in decreasing order of size actually inhibits bandwidth reduction: for starting vertex u=① and a corresponding LS(M) of width 11 ($T=-9$), the algorithm requires 56 vertex movements to yield an FLS of width 10 ($T=-7$). Although this result leads to an optimum numbering ($\delta=10$), much unnecessary work is done. Clearly, in order to evaluate the efficiency and utility of phase II of the SA algorithm, considerable computational experience, especially with large matrices, is desirable.

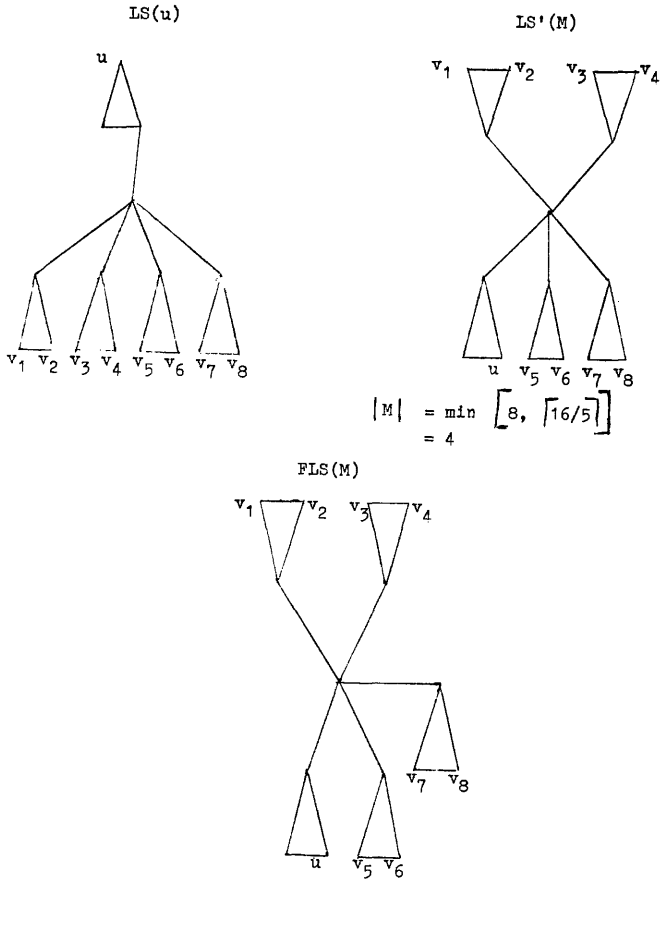

LS(u)
u
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8
LS'(M)
v_1 v_2 v_3 v_4
u v_5 v_6 v_7 v_8
$|M| = \min\left[8, \lceil 16/5 \rceil\right]$
$= 4$
FLS(M)
v_1 v_2 v_3 v_4
v_7 v_8
u v_5 v_6

Figure 4

PHASE III OF SA: NUMBER FLS

We propose here a somewhat more sophisticated numbering algorithm than has previously been employed,[5,7,9] but which retains the property of having execution time approximately linear in n. Indeed, as we shall see, the numbering algorithm has the interesting property of being rather more efficient in the more difficult cases.

Suppose we are given an FLS(M) of length λ and width w', with levels L_k' of width $w'(k)$, $k=1,2,\ldots,\lambda$. Suppose further that corresponding to each $v \epsilon V$, we may identify g_v, the level of v, and A(v), the set of vertices adjacent to v; and that corresponding to each level L_k' of FLS, we may identify the vertices $x_{kj} \epsilon L_k'$, $j=1,2,\ldots, w'(k)$. Phase III assigns numbers to the vertices on a level-by-level basis, with the objective of arriving at a numbering α of G which corresponds to a suitably small bandwidth δ_α. In the large, phase III proceeds as follows:

(1) $\Delta \leftarrow 0$.

(2) $\delta_\alpha \leftarrow w' + \Delta$.

(3) For each $k=1,2,\ldots,\lambda$, try to assign numbers to the vertices $x_{kj} \epsilon L_k'$ in such a manner that the *number* n(x) of each vertex x satisfies

$$|n(x) - n(y)| \leq \delta_\alpha,$$

for every $y \epsilon A_x$.

(4) If for some k, it turns out that the vertices cannot be numbered to satisfy this condition, then $\Delta \leftarrow \Delta + 1$, goto (2).

We observe that, in principle, this algorithm will terminate sooner or later; indeed, as CM remark,[5] for $\delta_\alpha = 2w'-1$, *any* level-by-level numbering of FLS will satisfy the condition given in step (3). We observe further that, in practice, the algorithm will normally terminate at some value δ_α close to w'; in fact, in the great majority of cases, for $\delta_\alpha = w'$.*

We give now a more detailed description of step (3). Since the same procedure is used for the numbering of each level, we confine ourselves to describing the numbering of the k^th level L_k'. Certain basic values need to be defined and calculated:

$$W'(k) = \Sigma_{1 \leq k' \leq k} w'(k'), \; 1 \leq k \leq \lambda;$$
$$= 0, \; k=0.$$

$A_{\pm 1}(j) = \{v \mid x_{kj} \epsilon L'_k \wedge v \epsilon A(x_{kj}) \wedge v \epsilon L_{k\pm 1}'\}$, the set of vertices of L'_{k+1} (L'_{k-1}) adjacent to the j^th vertex in L'_k.

* However an unpublished example due to Arany-Szóda has the following characteristics: $n = 60$, $m = 112$, $w' = 9$, $\delta(G) = 14$.

$a_{\pm1}(j) = |A_{\pm1}(j)|$.

$\text{lown}(j) = \min_{x \in A_{-1}(j)} n(x),\ a_{-1}(x) > 0;$
$\quad = W'(k-1),\ a_{-1}(x) = 0;$
the lowest number assigned to any vertex in L_{k-1}' which is adjacent to the j^{th} vertex in L_k'.

The processing required for each level falls naturally into two stages, which we call INITIALIZE L_k' and NUMBER L_k':

```
INITIALIZE $L_k'$
(1) For each vertex $x_{kj}$, $j=1,2,\ldots,w'(k)$, calculate
    (1.1) xmin(j)←max[$1+W'(k-1)$, $W'(k)+a_{+1}(j)-\delta_\alpha$], the least number assignable to $x_{kj}$
          which is compatible with bandwidth $\delta_\alpha$;
    (1.2) xmax(j)←min[$W'(k)$, $\delta_\alpha$+lown(j)], the greatest number assignable to $x_{kj}$ which is
          compatible with $\delta_\alpha$;
    (1.3) xrange(j)←xmax(j)−xmin(j)+1; if xrange(j)≤0, goto INCREMENT Δ (step (4) in the general
          outline of phase III given above).
(2) nrangeclass(j)←0, $j=1,2,\ldots,w'(k)$.
    [The number of vertices to which a number $W'(k-1)+j'$, $j'=1,2,\ldots,w'(k)$, may be assigned will
    be determined; nrangeclass(j) will then contain a pointer to the first of the numbers $W'(k-1)+j'$
    which may be assigned to exactly j vertices.]
(3) For each number $j'=1,2,\ldots,w'(k)$:
    (3.1) calculate
          nr(j'), the number of values of j for which
            xmin(j)≤j'≤xmax(j);
          nlist(j',1) to nlist(j',nr(j')), a list containing the values of j for which
            xmin(j)≤j'≤xmax(j);
    (3.2) if nr(j')=0, goto INCREMENT Δ;
    (3.3) np(j')←nrangeclass(nr(j')), nrangeclass(nr(j'))←j'.
          [np(j') is a pointer to the next number in nrangeclass(nr(j')).]
NUMBER $L_k'$
(4) J←1, counter←0.
(5) If nrangeclass(J)=0, J←J+1, goto (5).
    [The numbers to be assigned are chosen in increasing order of nrangeclass.]
(6) j'←nrangeclass(J), nrangeclass(J)←np(j').
    [j' is the number to be assigned.]
(7) [Determine j, the vertex to which j' is assigned.]
    (7.1) h←1, j←nlist(j',h);
    (7.2) h←h+1, $j_1$←nlist(j',h); if $j_1$=0,
          goto (7.6);
    (7.3) if range($j_1$)<xrange(j), xrange(j)←
          xrange(j)−1, j←$j_1$, goto (7.2);
    (7.4) xrange($j_1$)←xrange($j_1$)−1; if xrange
          ($j_1$)=0, goto INCREMENT Δ;
    (7.5) goto (7.2);
    (7.6) exit.
(8) [Assign j' to j.]
    (8.1) n($x_{kj}$)←$W'(k-1)$+j';
    (8.2) counter←counter+1; if counter=w'(k),
          goto (11);
    (8.3) xrange(j)←∞, nr(j')←0.
(9) [Delete j and j' from storage.] For every J'=xmin(j), xmin(j)+, ..., xmax(j) such that
    nr(J')=0, do the following:
    (9.1) rold←nr(J'), nr(J')←nr(J')−1;
    (9.2) if rold=1, goto INCREMENT Δ;
          if rold=J, J←nr(J');
    (9.3) pold←np(J'), np(J')←nrangeclass
          (nr(J')), nrangeclass(r(J'))←J';
    (9.4) if nrangeclass(rold)=J', nrangeclass(rold)
          ←pold, goto (9.7); otherwise,
          $j_2$←nrangeclass(rold);
    (9.5) if np($j_2$)=J', $j_2$←np($j_2$), goto (9.5);
    (9.6) np($j_2$)←pold;
    (9.7) exit.
(10) Goto (5).
(11) Exit.
```

Table II displays results obtained using the above algorithm on a few examples; these results [δ(SA)] are compared with optimum numbering [δ(FLS)] and with results [δ(GPS)] obtained by the GPS numbering algorithm. δ(FLS) was attained neither by SA nor GPS, primarily because more than single-level lookahead was required.

It appears that, in phase III as in the other phases, SA produces results at least as satisfactory as those of GPS. The execution time of phase III is bounded above by a value proportional to $\Sigma w'(k)^2$, and depends essentially on the size of the variables xrange(j) and nr(j'): when these variables are small—that is, when the numbering is more difficult—execution time will correspondingly be small. Storage required for phase III is of the order of $w'(w'+6)$.

CONCLUSIONS

We have described a bandwidth reduction algorithm which appears to be competitive in effectiveness and efficiency with presently known algorithm. Systematic testing on large matrices encountered in practice is re-

TABLE II—Numbering FLS using SA

FLS	δ(SA)	δ(FLS)	δ(GPS)
Figure 2	6	5	6
Figure 3	11	10	11
FLS of width 10 determined by applying phase II to Curtis matrix $[L_1' = \{52,53,54\}]$	11	10	12
Figure 4	5	4	6

quired. Some of the more important questions remaining to be clarified, either by experiment or by analysis, are as follows:

(1) The execution time of SB (phase I of SA) is proportional to 2Hm. What can be said about the magnitude of H?

(2) In what (more efficient) way can vertex movements be evaluated during phase II, in order to yield an optimum FLS?

(3) Given two FLS of G of widths w_1 and $w_2 > w_1$, denote by δ_1 and δ_2 the least bandwidths obtainable by level-by-level numbering of the first and second FLS, respectively. Does it follow that $\delta_2 \geq \delta_1$?

REFERENCES

1. Willoughby, Ralph A., *Sparse Matrix Algorithms and Their Relation to Problem Classes and Computer Architecture*, IBM Research Publication RC 2833, March 1970, 38 pp.
2. Pooch, Udo W. and Al Nieder, "A Survey of Indexing Techniques for Sparse Matrices," *ACM Computing Surveys* 5-2, June 1973, pp. 109-133.
3. Tewarson, R. P., "Computations with Sparse Matrices," *SIAM Review*, 12-4, October 1970, pp. 527-543.
4. Curtis, A. R. and J. K. Reid, "The Solution of Large Sparse Unsymmetric Systems of Linear Equations," *Proc. IFIP Congress 71*, 1972, pp. 1240-1245.
5. Cuthill, E. and J. McKee, "Reducing the Bandwidth of Sparse Symmetric Matrices," *Proc. 24th Nat. Conf. ACM*, 1969, pp. 157-172.
6. Rose, D. J., *Symmetric Elimination on Sparse Positive Definite Systems and the Potential Flow Network Problem.* Ph.D. thesis, Harvard Univ., 1972.
7. Arany, Ilona, W. F. Smyth and Lajos Szóda, "An Improved Method for Reducing the Bandwidth of Sparse Symmetric Matrices," *Proc. IFIP Congress 71*, 1972, pp. 1246-1250.
8. Smyth, W. F. and W. M. L. Benzi, "An Algorithm for Finding the Diameter of a Graph," *Proc. IFIP Congress 74*, 1975, pp. 500-503.
9. Gibbs, Norman E., William G. Poole and Paul K. Stockmeyer, *An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix*, Technical Report No. 5, July 1974, 25 pp.
10. Arany, Ilona and Lajos Szóda, "Ritka Szimmetrikus Matrixok Sávszélesség Redukciója, "*Információ Elektronika* 4, 1973, pp. 273-282.

Business opportunity analysis

JOHN P. DOLAN
Crown Zellerbach
San Francisco, California

and

by JAMES S. KETCHEL
University of Puget Sound
Tacoma, Washington

ABSTRACT

Opportunity analysis is proposed as a technological forecasting method for assessing monetary profit to a firm. The method can employ use of the Delphi committee, the Bayesian algorithm, computer conferencing, and impact analysis, with certain suggested modifications.

INTRODUCTION

Opportunity analysis involves the assessment, in terms of its monetary potential for profit, of any single or series of potential developments affecting the business. The potential developments could be a set of exogenous events, such as a reduction of taxes or the passage of favorable legislation. On the other hand, it can look at endogenous events such as a proposed cost reduction program. Finally, it can be used to evaluate such things as investment proposals which contain both internal and external variables. This paper presents the concepts underlying opportunity analysis and a suggested procedure for conducting an opportunity analysis.

UNDERLYING CONCEPTS

Opportunity analysis combines the Bayesian decision process with the Delphi method, with modifications discussed in this paper.

The Bayesian process[1] involves making a roster of future possible events; assigning a subjective probability to each event; making a list of alternative actions; setting a payoff for each matrix action cell; calculating the expected value for each action; and, finally, choosing that alternative with the highest value.

The Delphi method, as originally designed[2] and developed at the RAND Corporation,[3-7] is a technique for obtaining group opinion. The method has three features; opinions of group members are anonymous; group members have the opportunity to change individual opinions through iteration and feedback; and, group opinion emerges (and perhaps converges) as a statistical summary of individual responses. The Delphi method was designed to overcome the effect of individual decision maker bias that might be found in the Bayesian decision process. In implementing the Delphi method, the first step is to obtain a panel of experts to whom a coordinator mails a questionnaire. Anonymity of individual panel members' opinions is preserved as the questionnaire is returned directly to the coordinator. In the second step, the coordinator summarizes opinions and mails the summary to the panel for review. The process may be iterated as members review the responses of others and the summary causing a reconsideration of opinion. Finally, a group consensus unfolds as a statistical summary of opinion.

RESEARCH AND REFINEMENTS

Within the past decade, applications of Delphi have proliferated. To mention but a few, Delphi has been used to forecast data processing technology,[8] communications technology,[9] European political events,[10] food and nutrition technology,[11] automobile tire technology,[12] international affairs,[13] computer developments and applications,[14] biomedical research and drug therapy,[15] and weaponry technology.[16] Delphi has also been used for policy making. Applications have been in civil defense,[17] and in teacher education.[18] Concomitant with the applications research have been refinements to the Delphi method. Of particular interest to us in developing opportunity analysis have been four types of refinements:

(a) Use of the Bayesian process. It has been suggested[19] that probabilities be assigned to events. This refinement has been applied to assessing threats to an organization,[20] wherein it was felt that time periods are not necessarily mutually exclusive and that a particular event might occur in any of several time periods, each with its own probability, within the forecasting horizon.

(b) Use of the computer. The computer has been used to produce the statistical summaries of the Delphi iterations,[20] and has been used as the medium for transmitting questions to the panel and for receiving responses.[21,22]

(c) Use of a preliminary panel. The question has been raised[23] as to the source of the questions posed to the Delphi committee. One approach[20] is to have an exploratory conference of knowledgeable individuals who produce a grocery list of issues for subsequent consideration by the Delphi panel.

(d) Consideration of dependent or interrelated events. Referred to as cross correlations[24] and cross-impact analysis[10,25-28] this refinement to the Delphi method considers the likelihood of one particular event in the forecast, given the occurrence of one or more other forecasted events.

Opportunity analysis proposes to adopt these four refinements.

THE OPPORTUNITY ANALYSIS PROCESS

In developing business plans, one of the most difficult assessments is what projects or businesses a company should engage in. Commonly this work is handled by a new idea gathering unit of some type, or may be the result of approaches from outside the organization. In any case, a series of judgments must be made as to the viability and profitability of suggested new products, and this often is done for each project standing on its own.

Not uncommonly, related projects occur at one or more research locations remote from the corporate headquarters, and so the researchers and corporate staff personnel operate without benefit of each other's insight. This is particularly costly to the problem of project selection and new business development. An alternative to the deterministic discounted cash flow capital budgeting approaches is opportunity analysis. This is a five part process which utilizes the principles of the Delphi process and Bayesian algorithm as outlined previously.

Step One. Idea input.—During this stage of the process, there would be a general survey of the members of the corporation dealing with technology development. The survey can be conducted by mail or by conferencing on a computer terminal. The important element is that the technology be spelled out in terms of what need it fulfills, i.e., savings in unit labor costs, decreased raw materials cost, increased performance, the estimated cost of commercializing the technology, and the length of time it is expected to take to achieve it.

At this phase of the investigation it is best not to identify the donor in order to help overcome the "not invented here" problem. It is also important that all offerings are invited without imposing criteria so that a free flow is inspired.

This approach is contrasted with the method employed in[20] where threats to the organization were identified as a result of a series of face-to-face committee meetings.

Step Two: Idea exchange.—From the array of inputs is assembled a master list of the developments, without the time or money figures shown. This list is then fed back to the group, via mail or computer conferencing, for estimation of what ideas would be likely to complement others, and what would be likely to supplant others. For each overlap area, the degree of overlap would be estimated. This degree of overlap is used to establish cross-impacts from one idea to another. This is necessary so that later in the process we can establish the total impact of an event across time, rather than its simple, immediate effect.

Step Three: Critical factor analysis.—Each group of ideas is tested for cross correlation and cross-impact. We view cross correlation analysis being appropriate for two simultaneous and related events. By contrast, we view cross-impact analysis being used for the circumstance of temporal priority when one event (or more than one event) precedes and is a prerequisite to another event. When either of the above conditions is found, a private meeting is held between the coordinator and each panel member internal to the organization to probe the key assumptions underlying the ideas. In the case of the labor saving device, this might be that wage rates will rise. For increased performance, the assumption may be that the result will be a higher price for the finished product. The factors which could change and impact these assumptions must be probed. For the critical factor analysis to be effective, the panel member should sort out in his/her mind whether events are dependent or related.

Once this is done, a series of testable statements is made by the coordinator which relate to the factors. In the case of a labor saving device, E_{11} (See Table I) this might read "Labor rates for this class of labor will rise six percent per year" E_{12}. As a statement measuring the overall effect, we would also need to evaluate the event "Labor rate adjustment by union contract reflecting six percent productivity increase" E_{13}. For the product with improved performance E_{21}, we might say "A product with this feature can be marketed at a $5.00 premium to current competitive items" E_{23}. A related event would be "Introduction of a competitor's product with this feature" E_{22}. These events must be clear enough to allow an evaluation of their probability of being true.

TABLE I

	Unconditional Probability	Cross Correlation	Cross Impact
Event Set 1: Labor Saving Device	E_{11}	E_{12}	E_{13}
Event Set 2: Product Improvement	E_{21}	E_{22}	E_{23}

Further, we should point out that it is possible for E_{23} and E_{12} to be cross-impacted, given that the union might ask for a wage increase due to larger business revenues.

The analyst might want to plot the probabilities for each sample space above, as illustrated in Figure 1, to show an increasing probability over time.

Step Four: Likelihood assessment.—For each of the statements made a suitable group of experts, most of whom would be external to the firm, is selected capable of weighing the probability of occurrence. For the labor-saving device, this might be labor economists or personnel specialists. For the product whose performance is enhanced, this could be a sales group or a panel of consumers of the product. These should be people knowledgeable in the area; seeking a single "expert" is to be avoided. It will be necessary to employ the Delphi procedure of resubmitting those events which do not achieve a reasonable consensus on the first pass.

Step Five: Opportunities Matrix.—The assessment is made in terms of probability of occurrence within selected time frames. For each time frame, the Bayesian algorithm is applied by multiplying the probability times the dollar value of the event. The details for this procedure have been given in Reference 29.

The summary of the above steps produces a final product grouping which shows the collected ideas, their estimated costs and benefits, and a probability of the revenues from those benefits being achieved. This is arrayed from top down, with the package of ideas having the highest probabilistic revenue relative to costs (from the Bayesian algorithm) on top and downward to those with lower and perhaps negative returns. With such a list in hand, it is much simpler for management to probe for new product ideas, and to discuss with those in research those areas with the highest promised returns. It may also be possible to discuss, using the earlier estimates, the possible tradeoff of additional expenditures on favorable projects in order to reduce the time necessary to produce the benefits.

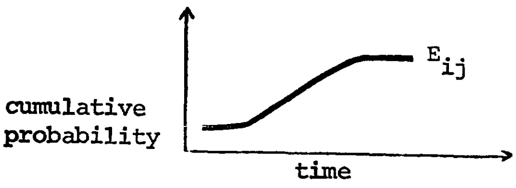
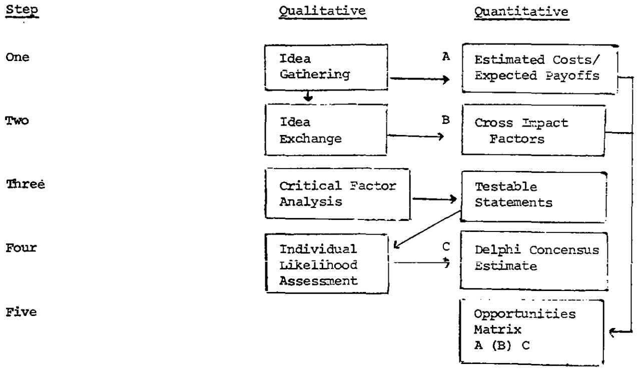


Figure 1

The five steps could be represented as a flow chart:

OPPORTUNITY ANALYSIS PROCESS



SUMMARY

We have outlined a procedure for assessing in monetary terms the likely value of an innovation or group of innovations to a corporation. This is done using the principles of the Bayesian algorithm and the [illegible] technique, as modified. The extension of the procedures from those used in impact analysis[20] has required certain changes which are discussed.

ACKNOWLEDGMENT

The authors thank Dr. Mitchel F. Bloom, University of Puget Sound, for his review and constructive critique of the original draft of this paper.

REFERENCES

1. Bierman, Harold, Jr., Charles P. Bonini and Warren H. Hausman, *Quantitative Analysis for Business Decisions,* 4th ed. Homewood, Illinois; Richard D. Irwin, Inc., 1973, pp. 68-77.
2. Helmer, Olaf and Norman Dalkey, "An Experimental Application of the Delphi Method to the Use of Experts," *Management Science,* Volume 9, Number 3, April, 1963.
3. Gordon, T. and O. Helmer, *Report on a Long-Range Forecasting Study,* The RAND Corporation, P-2982, September 1964.
4. Brown, Bernice and Olaf Helmer, *Improving the Reliability of Estimates Obtained from a Consensus of Experts,* Report P-2986, The RAND Corporation, September, 1964.
5. Dalkey, Norman C., *The Delphi Method: An Experimental Study on Group Opinion,* The RAND Corporation, RM-5888-PR, June, 1969.
6. Dalkey, Norman C., B. Brown and S. Cochran, *The Delphi Method, IV: Effect of Percentile Feedback and Feed-in of Relevant Facts,* The RAND Corporation, RM-6118-PR, March, 1970.
7. Dalkey, Norman C. and Daniel L. Rourke, *Experimental*

Assessment of Delphi Procedures with Group Value Judgments, The RAND Corporation, R-612-ARPA, February, 1971.

8. Cetron, Marvin J., *Technological Forecasting,* New York, Technology Forecasting Institute, 1969, pp. 145-160.
9. Sulc, O., "A Methodological Approach to the Integration of Technological and Social Forecasts," *Technological Forecasting,* Volume 1, 1969, pp. 105-108.
10. Enzer, Selwyn, "Delphi and Cross-Impact Techniques: An Effective Combination for Systematic Futures Analysis," *Proceedings of the International Future Research Conference,* Vol. 1, Kyoto, Japan, April, 1970.
11. Wooden, Robert P. and Bill R. Richeson, "Technological Forecasting: The Delphi Technique," *Food Technology,* Vol. 25, October, 1971, pp. 59-63.
12. Kovac, Frederick J., "Technology Forecasting—Tires," *Chemical Technology,* January, 1971, pp. 18-23.
13. Martino, Joseph P., "An Experiment with the Delphi Procedure for Long-Range Forecasting," *IEEE Transactions on Engneering Management,* Vol. EM-15, No. 3, September, 1968, pp. 138-144.
14. Bjerrum, Chresten A., "Forecast of Computer Developments and Applications 1968-2000," *Futures,* Vol. 1, No. 4, June 1969, pp. 331-338.
15. Bender, A. Douglas, Alvin E. Strack, George W. Ebright and George von Haunalter, "Delphic Study Examines Developments in Medicine," *Futures,* Vol. 1, No. 4, June, 1969, pp. 289-302.
16. Cetron, Marvin J. and D. N. Dick, "Producing the First Navy Technological Forecast," *Technological Forecasting,* Vol. 1, 1969, pp. 185-195.
17. Turoff, Murray, "The Design of a Policy Delphi," *Technological Forecasting and Social Change,* Vol. 2, 1970, pp. 149-171.
18. Cyphert, Frederick R. and Walter L. Gant, "The Delphi Technique: A Tool for Collecting Opinions in Teacher Education," *The Journal of Teacher Education,* Vol. 21, No. 3, Fall, 1970, pp. 417-425.
19. Blackman, A. Wade, Jr., "The Use of Bayesian Techniques in Delphi Forecasts," *Technological Forecasting and Social Change* 2, 1971, pp. 261-268.
20. Ketchel, James S. and John Dolan, "Impact Analysis," *Proceedings of the Association for Computing Machinery Annal Conference,* San Diego, California, November, 1974, pp. 318-325.
21. Turoff, Murray, "Delphi Conferencing: Computer-Based Conferencing with Anonymity," *Technological Forecasting and Social Change,* Volume 3, Number 2, 1972, pp. 159.
22. Turoff, Murray, "Conferencing via Computer," Information Networks Conference, 1972, pp. 194-197.
23. Martino, Joseph P., *Technological Forecasting for Decisionmaking,* New York, American Elsevier Publishing Company, Inc., 1972, p. 26.
24. Gordon, T. J., "New Approaches to Delphi," in *Technological Forecasting for Industry and Government,* edited by James R. Bright, Englewood Cliffs, New Jersey, Prentice-Hall, Inc., 1968, pp. 134-143.
25. Gordon, T. J. and H. Hayward, "Initial Experiments with the Cross Impact Matrix Method of Forecasting," *Futures,* Vol. 1, No. 2, December, 1968, pp. 100-116.
26. Turoff, Murray, "An Alternative Approach to Cross Impact Analysis," *Technological Forecasting and Social Change,* Vol. 3, 1972, pp. 309-339.
27. Wilson, Ian H., "Socio-Political Forecasting: A New Dimension to Strategic Planning," *Michigan Business Review,* July, 1974, pp. 15-25.
28. Bloom, Mitchel F., "Deterministic Trend Cross-Impact Forecasting," *Technological Forecasting and Social Change,* Vol. 8, 1975, pp. 35-74.
29. Dolan, John and James S. Ketchel, "Stochastic Risk Analysis in Capital Budgeting," joint meeting of the Operations Research Society of America and the Institute of Management Science, San Juan, Puerto Rico, October, 1974.
30. Linstone, Harold A. and Murray Turoff (editors), *The Delphi Method,* Reading, Massachusetts, Addison-Wesley Publishing Company, 1975, p. 3.

A "unique number" generator

by ARMEN NAHAPETIAN
Arya-Mehr University of Technology
Tehran, Iran

ABSTRACT

In many computer systems and applications, the need arises for the software to generate unique numbers or names for the identification of dynamically generated entities.

Such a generator is described here which produces new numbers and reuses numbers released by the system.

An analysis is carried out to determine the storage requirements of the generator in relation to the number of requests that can be accommodated.

INTRODUCTION

In computer applications, situations often arise where the software dynamically generates entities that must be named for future reference.

Examples of such cases are compilers that generate names for temporary locations in assembly language, graphic systems that generate names for created subpictures, operating systems that generate internal names for jobs submitted or files created, etc.

The generation of unique names for such cases can, of course, be accomplished by incrementing a counter each time a new name is requested. The resulting integer can directly be used as a name or converted to an appropriate base if an alphabetic or alphanumeric name is needed.

For example, to generate five-character Fortran identifiers, the number can be converted to a mixed radix integer

$$a_{26}\, b_{36}\, c_{36}\, d_{36}\, e_{36},$$

the first digit representing a letter and the rest alphanumeric characters.

The method just described has the following drawback. Since in most applications the names generated are used for a period of time and then released; eventually the counter will be incremented to its limit, even if the set of names currently in use has a much smaller size than this limit.

With additional restrictions imposed on the names, the limit of the counter might be reached even sooner. If, for example, names were restricted to one letter followed by one digit, this limit would be 260.

To somewhat overcome this inefficiency, the number generator to be described can be used which not only generates new names, but also tries to reuse as many of the names released as possible.

THE GENERATOR

The generator works by simply storing the released numbers (names) in a fixed-size stack. If a new number is requested the stack is tried first and only in case of an empty stack is the counter incremented to generate a new number.

Since the size of the stack is finite, cases might arise where a returned number cannot be pushed onto a full stack. Thus, the reuse of such numbers is eliminated. The results obtained in section three can be used for choosing an appropriate stack size based on the counter limit and the total number of requests anticipated.

The algorithms that follow represent the "request" and "return" operations for the unique number n. The initial value of the variable "pointer" should be zero and "counter" must initially be set to the smallest integer that can be represented or is appropriate for a name. "Stacksize" and "countlimit" represent the maximum capacity of the stack and the maximum count of the counter, respectively. The array "stack" and integers "pointer," "countlimit" and "stacksize" are assumed to be declared globally.

```
procedure request (n);
integer n;
Begin
  if pointer>0 then begin n←stack (pointer); pointer
                 ←pointer-1 end
                 else if counter=count-
                 limit then overflow
                        else begin
                               counter←counter+1;
                               n←counter
                               end
end request;
```

```
procedure return (n);
integer n;
```

```
Begin
  if pointer < stacksize then begin
                              pointer ← pointer + 1;
                              stack (pointer) ← n
                              end
end return;
```

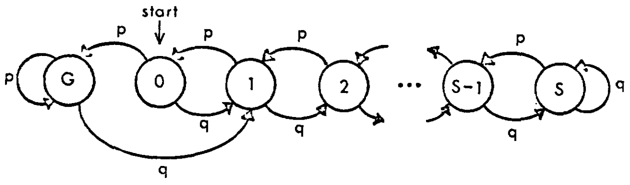
THE STACK SIZE

To determine the minimum stack size for a specific application, in addition to the number of names needed and the counter capacity, statistical information on the occurrence of the requests and returns must be provided. Of course in any case the stack size need not be larger than the number of distinct values the counter can produce.

The results of the following analysis can be used in cases where the occurrence of requests and returns (transitions) can assume to be a Markov process,[1] i.e., the probability of a transition being a request or a return is independent of previous transitions. The results can even be used in applications where a large number of request lead into a steady state Markov process, and a large number of returns terminate the process. Since to determine the stack size in such a case one need only consider the steady state (at the beginning of the process the stack is almost empty, and at the end, most numbers lost due to a full stack will not be needed again for reuse).

Assuming a probability p for a request and $q=1-p$ for a return at each transition, the state diagram and the transition probability matrix of the process are shown in Figure 1.

The number representing each state is the number of elements in the stack of size S and the state represented by G is the state where each time entered (due to a request) a new number must be generated by the counter due to an empty stack.



$$\begin{array}{c|cccccccc}
 & G & 0 & 1 & 2 & & & S-1 & S \\
\hline
G & p & 0 & q & 0 & \cdots & 0 & 0 & 0 \\
0 & p & 0 & q & 0 & \cdots & 0 & 0 & 0 \\
1 & 0 & p & 0 & q & \cdots & 0 & 0 & 0 \\
\vdots & & \vdots & & & & & \vdots & \\
S-1 & 0 & 0 & 0 & 0 & \cdots & p & 0 & q \\
S & 0 & 0 & 0 & 0 & \cdots & 0 & p & q
\end{array}$$

Figure 1

Using T_i to represent the expected number of transitions needed to reach state G from state i, and assuming C to be the number of distinct values the counter can produce, the expected number of transitions for the generator to overflow can be shown as

$$T_{over}=T_0+CT_G$$

i.e., the expected number of transitions for entering state G, $C+1$ times starting from state zero.

And the expected number of transitions before overflow occurs, being one less than T_{over} is

$$T=T_0+CT_G-1$$

T_0 and T_G can be obtained from the following set of equations

$$\begin{aligned}
&T_G=p+q(1+T_1)\\
&T_0=p+q(1+T_1)\\
&T_1=p(1+T_0)+q(1+T_2)\\
&T_2=p(1+T_1)+p(1+T_3)\\
&\cdot\\
&\cdot\\
&\cdot\\
&T_{S-1}=p(1+T_{S-2})+q(1+T_S)\\
&T_S=p(1+T_{S-1})+q(1+T_S)
\end{aligned}$$

to be

$$T_G=T_0=(1/p)\,(1+a+a^2+\ .\ .\ .\ a^S)=(1/p)\,(a^{S+1}-1)/(a-1)$$

where $a=q/p$, thus the expected number of transitions before overflow occurs is

$$T=(C+1)\,(1/p)\,(a^{S+1}-1)/(a-1)-1$$

And finally the expected number of requests that can be accommodated with a stack size of S and counter capacity of C,

$$R=pT=(C+1)\,(a^{S+1}-1)/(a-1)-p$$

which for $p=q=1/2$ reduces to

$$R=(C+1)\,(S+1)-1/2$$

CONCLUSION

A number generator was introduced that issues a unique number each time a request is made, trying to save and reissue returned number by using a stack.

The generator can be used for generating alphanumeric names by converting the generated numbers to an appropriate base.

Assuming the request and return transitions to be a Markov process, formulas were derived for the expected number of requests that can be accommodated in terms of the maximum capacities of the counter and the stack.

REFERENCE

1. Parzen, *Modern Probability Theory and its Applications,* Wiley, 1960.

An on-line test program for peripheral devices

by AKIRA TANEDA, HIKARU OKU and DAIJI NAMBA
Nippon Telegraph and Telephone Public Corporation
Yokosuka, Japan

ABSTRACT

An on-line test program is a program to detect troubles in devices being used in service or to assure correctness of their operation. Therefore, it is required that the on-line test program is able to safely certify device functions without any disturbance to the service. That is, since the on-line test program interrupts usual guards of an operating system, it may cause disturbances to the service, for example, by breaking contents of media used in the service. Therefore, the requirement to the on-line test program is not only to satisfy test functions but also to remove each disturbance cause which seems to affect the service. In addition, it is necessary to test automatically to improve the capability of the program and to avoid disturbances resulting from misoperations.

This paper describes design objectives, service interruption countermeasures and automatic test techniques for an on-line test program designed mainly to test peripheral devices of DIPS (Dendenkosha Information Processing System), that is a standard data communication system in NTTPC (Nippon Telegraph and Telephone Public Corporation).

The fault detection ability of this program was confirmed at a satisfactory level, according to experiments involving artificial device troubles.

INTRODUCTION

Information processing devices carry out important missions in every social activity as well as economic activity. Therefore, it is very important to assure their reliability. Especially, higher reliability is demanded in data communication systems offering on-line services.

It is natural that a system down condition or information destruction resulting from device troubles, would cause serious damages, not only to a specific user but also to every user. Therefore, generally, device trouble detection, trouble influence prevention and trouble recovery are taken into consideration by means of countermeasures involving redundant design in a device, multiple installation of the same devices, fail soft control and so on, in a system design stage of a data communication system.

On the other hand, after the system is operational, periodic inspections and adjustments of devices to reduce the mean time between failures (MTBF) and rapid repairs to shorten the mean time to repair (MTTR) are required as maintenance operations.

In these maintenance operations, it is necessary to certify device functions after inspections and adjustments, to gather failure information about troubled devices, to locate failures and to certify device functions after repairs. Tools for these confirmations are maintenance panels, test equipment, test programs and fault locating programs. Each has individual characteristics and application fields, so they are not able to be compared simply with each other. However, on-line test programs, whose handling is easy and which do not require any special hardware for confirmation, are the most suitable for certifying device functions with the service in a short time without involving manpower.

It is not only necessary to satisfy test functions, but also to avoid disturbances to the service, because on-line test programs access together with service to devices to be tested.

The authors developed an on-line test program in DIPS (Dendenkosha Information Processing System) that is a standard data communication system in NTTPC (Nippon Telegraph and Telephone Public Corporation).

This program mainly tests functions of peripheral devices (file memories and low speed input output devices), under the configuration shown in Figure 1 for example. The reason why peripheral devices are selected is that peripheral devices have mechanical parts, so they cause trouble frequently and require many inspections and adjustments, as well as there being many devices in a system. In an example of a data communication system in NTT, maintenance tasks for peripheral devices run into 70-80 percent of all maintenance tasks in a system.

The fault detection coverage of this program is about 80 percent of all circuits in the devices according to the experiment by means of artificially induced troubles.

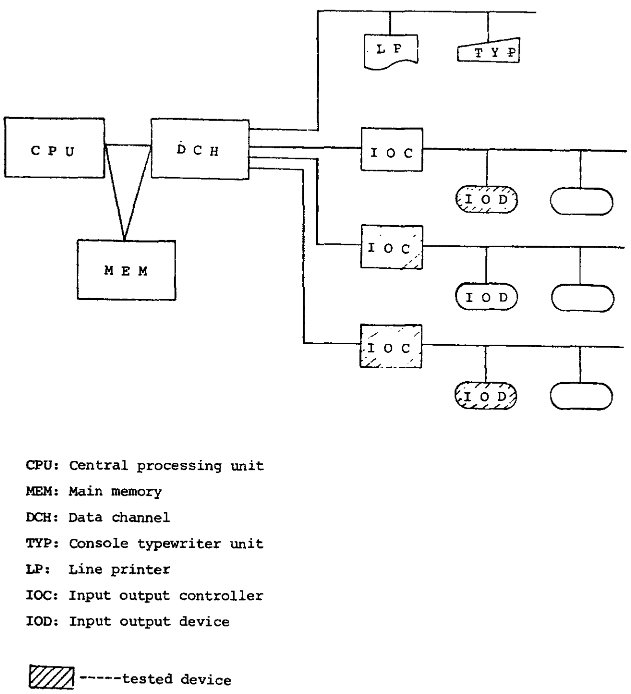


Figure 1—Test system configuration (example)

Design objectives, service interruption and misoperation prevention measures and evaluation of the program are covered in the following.

DESIGN OBJECTIVES

The following main design objectives are set for the test program:

(1) Making detailed tests
(2) Preventing service interruptions
(3) Making operation easy

(1) Making detailed tests—Generally, devices must handle many status transitions and many series of output patterns corresponding to many series of input patterns. Testing a certain function perfectly is confirmation of all status transitions and series of outputs corresponding to all inputs. It is impossible technically and economically to perform such tests for all functions of a device by program. Thus, the test program is not essentially perfect. Therefore, an important objective is how to increase the fault detection ability of the test program within technical and economical feasibilities by making detailed test items. To accomplish this objective, it is necessary to inspect a status transition and a series of outputs corresponding to at least a series of inputs for all functions which can be tested by the program.

(2) Preventing service interruptions—The test program interrupts usual guards for testing of an operating system. For example, it accesses troubled devices or uses inhibited commands to usual users. Therefore, running of the test program may seriously disturb the system service. It is necessary to try to find the cause of disturbances resulting from running the test program and to take the most suitable measures to meet any problems encountered.

(3) Making operation easy—There are many kinds of peripheral devices in a system. Even if variations of a kind of device are neglected by an operating system, the test program is aware of their varieties in many cases. In addition, it is necessary to select the most suitable program corresponding to status data tables of the operating system for devices to determine the test partition.

It is difficult for an operator to be aware of the varieties and to select the most suitable test program; in addition, he may cause disturbances resulting from misoperations. Therefore, test automation, such that a test program corresponding to these test conditions being automatically scheduled and executed, is required.

Ways to achieve these design objectives are mentioned in the following.

WAYS TO SATISFY TEST FUNCTIONS

As mentioned previously, it is necessary to test all device functions by using at least one series of inputs corresponding to each function. It is impossible to perform tests using all series of inputs.

Therefore, it is desired to test at least all functions prescribed by the logic specification of the device, by using one very severe data pattern for devices corresponding to each function. For example, when mechanical parts are tested, they shall be operated, insofar as possible, within the boundaries of their performance. When logical parts are tested, it is necessary that tested circuits be covered in more detail by being tested repeatedly with random data patterns.

SERVICE INTERRUPTION PREVENTION

Service interruption prevention is the most important problem in the design of an on-line test program. Causes which are considered to disturb the service are shown in Table I. Appropriate actions are required to remove these causes.

Incorrect access to a troubled device

In accessing to a troubled device by the on-line test program, there is a possibility of destroying the written information on the media of a correctly operating

TABLE I—Interruption Causes

ITEM	OUTLINE
Incorrect access to nomal devices	Destruction of contents in media in the service resulting from misoperations of troubled tested devices.
Garbage	Test trace which may remain after test program execution is stopped by some reasons.
Conflict with the service	Conflict with the service about system resouces which a test program is going to use.

device caused by incorrect action to it. This probability is very small. However, once incorrect access happens, the disturbance may cause widespread deleterious results on the service. Some examples of countermeasures are:

(1) In case of channel (CH) or input output controller (IOC) trouble, there is a possibility that input output operation of a testing system might cause incorrect access to an input output device (IOD), which doesn't belong to the testing system, by means of the incorrect behavior of CH, IOC and to destroy the recorded data in the media used in the service.

When the accessing route to the device consists of a dual path, assigning one path for the test and the other path for the service, it is possible to prevent trouble as shown in Figure 2. In this case, IODs in the service are reserved by using the device reserve command via the path for the service.

Functions of this command are to tie a specific IOD to a specific path. Namely, by tying the devices to the path for the service, each device for the service is free from incorrect access by the tested device.

(2) In the case of the band selector trouble in a magnetic drum memory, there is a possibility that another band could be accessed instead of the allocated band for the test and the recorded information on the correct band might be destroyed.

Accordingly it is necessary that band address information for the test must be fully checked before the test is started by the command.

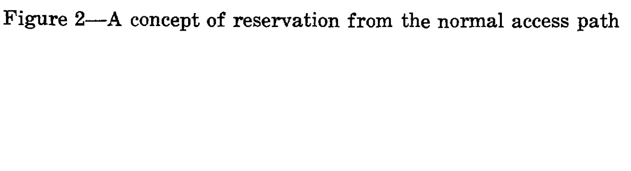


Figure 2—A concept of reservation from the normal access path

Garbage prevention

Garbage is defined as traces that may be left when running a test program is completed or interrupted. Two types of garbage are considered. One is the trace left on the control table of the operating system, for example, the discrimination flag to indicate whether each device is on test or not. The other is the trace left on the hardware of each device for example, the repeated status of the command. These traces must be cleared after the test is finished, because, later on, they will become causes of service disturbance. When a test is stopped due to the system being down, these traces are cleared at the time of restart of the operating system. However, when the test is stopped by operator command and abort is caused, the traces must be cleared by the test program itself. Moreover, in order to double the guard against service disturbance, it must have the function of clearing garbage by operator command.

Preventing service conflict concerning resources

There are conflicts with the service concerning all the system resources needed for the test.

To prevent the system from serious disturbance caused by interruption of the service by this conflict, the resources used in time sharing mode or space sharing mode, such as CPU or main memory, are to be owned by both the test system and the service. Resources which are difficult to hold in common (such as the device being tested) are owned exclusively by the test system.

Holding resources in common or exclusively for the test program results in lowering the processing power of the system. In short, it increases the overhead. Therefore, fundamental methods to deal with overhead are to shorten the testing time and to decrease the amount of resources needed for the test. They are as follows.

(1) In regard to CPU overhead, most of the test transaction is related to input-output operation, due to the nature of the test program, and the effect on CPU time is very small.

But, as some trouble might arise when the system is busy furnishing the service, a restriction must be imposed wherein a test program can't be used at such a time. Moreover, setting the test program as a background job for the service, the service should be done primarily.

By these means, overhead to CPU time for service can be made almost negligible.

(2) By avoiding the exclusive possession of devices necessary for the test insofar as possible and holding

them in common with service system, effective usage of resources can be accomplished.

However, in case the resources must be possessed exclusively due to test necessity, in order to prevent occupation for resources for a long time by hardware errors or program bug, it is necessary to watch the time while resources are used for the test. After a limited time is over, the test should be stopped, and resources must be returned to the service. Also the quantity of main memory must be minimized.

(3) The time for testing each device is usually limited to within 20 minutes. This is decided by considering avoiding exclusive possession of system resources for a long time and considering the necessary time for maintaining each device.

Accordingly some tests which need a long time to run (for example, the test for initializing the media) must be omitted, except in the case when a special order by the operator is being accomplished.

Considering the measures mentioned above, the time for the test could be shortened and system resource availability increased. After all, overhead to the service could be decreased.

OPERATION SIMPLIFICATION AND MISOPERATION PREVENTION

If an operator instigates a test which may destroy information on the active device by overwriting, for example, it may cause an extremely adverse disturbance to the service where the program runs as it is. Because a man sometimes commits careless mistakes, it is necessary to run the program as automatically as possible. When the operator intervenes, simple operation and prevention against misoperation are essential.

Test automation

For test automation, the test program should start automatically, through the detection of the trouble device by the operating system without the intervention of an operator. However, as explained in the previous section, it cannot be uniquely decided as to whether the system should be tested during the most busy time or not, and whether a long test should be performed or not. Therefore, it was decided that the program should start according to the start command by an operator.

Items that should be decided by the operator's judgment for test program execution are the tested device address number, execution of a test requiring long running time and so on.

Necessary information to select optimal test programs is classified according to the following attributes.

Attribute 1: noted on the peculiarity in running the test (whether operator intervention is necessary or not, whether the running time is long or short, and so on).

Attribute 2: based on the restriction of the I/O macro command (logical state of a tested device and its access path, whether the inhibition of the interruption is necessary or not and so on).

Attribute 3: media for tests noted (whether the volume for maintenance is necessary or not).

Attribute 4: based on peculiar conditions of the device (kinds of devices and so on).

TABLE II—Block Select Pattern

BIT POSITION	O B S P	A B S P
b_0	Block class is A	Block class A is able to run
b_1	Block class is B	Block class B is able to run
b_2	Block class is C	Block class C is able to run
b_3	Block class is D	Block class D is able to run
b_4	Block class is E	Block class E is able to run
b_5	Block class is F	Block class F is able to run
b_6 - b_{15}	Another block attribute	Parameters designated by an operator
b_{16} - b_{31}	Hardware inherent conditions corresponding to the test item	Hardware inherent conditions corresponding to designated device.

Some test items gathered due to their attributes are called Block.

A combination of these attributes decides the test items running condition (test condition). This test condition is prepared beforehand as a kind of data, called the Original Block Selected Pattern (OBSP), for every proper test item.

On the other hand, when the test running command is input, the parameters of that command and the condition of the tested devices decide the condition of the test item which can be executed. This is called the Actual Block Select Pattern (ABSP). A part of OBSP and ABSP is shown in Table II. The meaning of the block class (A—F) in the table is shown in Table III.

TABLE III—Block Class

Test condition \ Test item	Test not required interruption mask		Test required interrup tion mask*
	Test not occupying a channel	Test occupying a channel**	
Standard	F	D	B
Optional	E	C	A

* Test which occupies CPU during testing time

** Test which occupies channel during testing time

The program selects the only block which has the OBSP corresponding to ABSP.

According to such an automatic selecting function of the test items, an operator can test safely without being concerned with differences in the test programs, the states of tested devices, the kinds of devices and so on.

Protection against misoperation

Misoperation that causes extremely adverse effects to the service, include incorrect appointment of the tested device, incorrect mounting of the medium and the wrong establishment of the pseudo-trouble, for example.

For these misoperations, the following countermeasures apply.

(1) For the wrong appointment of the tested devices, it is sufficient to limit the test to the active device unless the area for test is prepared in that media.

(2) For incorrect mounting of a data storage media, a usual countermeasure is to use a specific medium for test in order to protect the media for service. Therefore, when the medium for test is not mounted, the program notifies the operator before it permits execution of the test.

(3) The test by pseudo-trouble is required to test the hard-core, such as the trouble detecting circuit and the trouble diagnosing circuit.

Misoperation in such a test is extremely adverse to the system and is not guarded by the program. Therefore, at least such a test should not be performed under the operating system in service.

Even if it can't satisfy the necessary condition, it is thought inevitable to cancel some test items in order to avoid the disturbance. By following such procedures, the safety of the test will be increased much more.

PROGRAM STRUCTURE

The structure of this test program is shown in Figure 3.

The test control part, the part of controlling and helping test execution, mainly carries out the function shown in Table IV.

The test execution part covers whole necessary test items.

EVALUATION

In previous sections, design objectives and implementation are described.

Performance of the test program and its influence on service by the test execution will be described in this section.

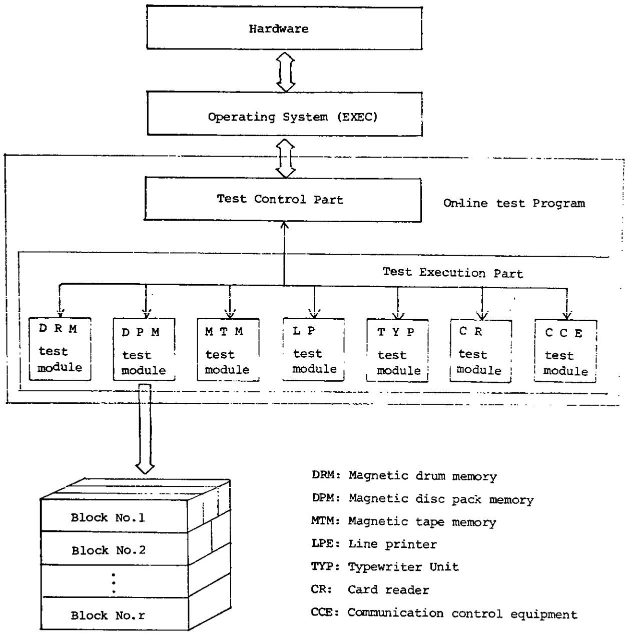


Figure 3—On-line test program structure

Performance

Incompleteness of testing functions is caused by selecting only those test items which can be executed automatically. In addition, tests using perfect data patterns are not able to be implemented. In relation to this problem, an example of results of experiments using pseudo-troubles is described for a magnetic disc pack memory (DPM) and a lineprinter (LP) in DIPS.

TABLE IV—Summary of Test Control Part Function

No.	FUNCTION OF TEST CONTROL PART
1	automatic running control
2	command processing
3	task control
4	event processing
5	prevention disturbance
6	setting up test conditions
7	resource management
8	macro processing

(1) Experimental method and results—Pseudo-troubles are made to stick logic signals at '0' or '1' at the terminals of connectors for logic circuits packages. This makes single and solid trouble.

Several hundreds of troubles are set, selected at random from logic diagrams of the devices. The test program performs each test for the pseudo-troubles and is checked to determine whether it detects pseudo-troubles or not.

Experimental results are shown in Table V.

(2) Consideration—Circuits that can't be activated by the program as shown in Table V are circuits related to the maintenance panel and so on, whose troubles can't be detected by any programs. Therefore, problems are in parts where the pseudo-troubled circuits were justified normal in Table V. These parts consist of (A) parts that can't be tested automatically by the program and (B) parts that can't be tested as a result of incompleteness of data patterns.

Since (A) parts are mainly hard-core for the diagnosis and their test requires setting pseudo-troubles, the test automation can't be implemented.

These parts form about 10 percent of all circuits in DPM and about three percent in LP.

On the other hand, about 17 percent of pseudo-troubled circuits in DPM and about four percent in LP were justified as shown in Table V.

Therefore, it is considered that the remaining seven percent in DPM and one percent in LP belong to (B). As mentioned above, it is almost impossible technically and economically for these parts to be perfectly covered. Thus it is considered that this amount of defects in fault detection is inevitable.

Defects in fault detection caused by (A) are mostly related to hard-core prepared for fault locating program. Such a hard-core is not used by the operating system or the service, so it is not essential to test them.

It was confirmed that the test program described in this paper can satisfy more than 90 percent of trouble detection ability.

TABLE V—Experiment Result Due to Pseudo-trouble

DEVICE	NUMBER OF PSEUDO-TROUBLE POINTS	PERCENTAGE OF TROUBLE DETECTION	PERCENTAGE OF TROUBLES NOT ABLE TO BE DETECTED BY ANY PROGRAM	PERCENTAGE OF TROUBLES JUSTIFIED NOMAL BY THE TEST PROGRAM
DPM	311	72%	11%	17%
LP	213	83%	13%	4%

Test influence on service

There are test items that have to occupy an access path (channel, IOC) to accomplish a test, for example the test to confirm whether a channel is busy. In the case of occupation of one of the dual paths for a test, if some troubles occur at the same time at the other of the dual paths, service for some file users connected to this path becomes impossible. When a single path is connected to some on-line devices, the path occupying test is not performed. In the case of occupation of one of the dual paths for a test, probability P_f for the other path (CH,IOC) to encounter a trouble is given by

$$P_f = 1 - e^{-\alpha t/tm}$$

$tm = 10^9/f_{ch.ioc}$ $f_{ch.ioc}$; FITs
where t; test execution time of the path
α; path occupying rate tm; MTBF

α is about 20 percent, if t is one hour per day then P_f is about 0.002.

This means that trouble in the other path during the test execution occurs once every five thousand days. Therefore, it is clear that this probability P_f is negligible.

CONCLUSION

Design objectives, methods to realize them and evaluation about the on line test program have been mentioned.

It was shown that functions required for testing devices were satisfied by using at least one severe data pattern.

Disturbances caused by devices were mentioned along with countermeasures preventing disturbances to the service, and those caused by men were discussed along with countermeasures preventing misoperations. So, it became clear that the test programs, produced by ways mentioned in this paper are able to be utilized effectively and had little adverse influence on service. These test programs are expected to allow device maintenance to become easier and the cost of maintenance will become lower by its usage.

Structure of the ELF operating system*

by DAVID L. RETZ
Stanford Research Institute
Menlo Park, California

and

BRUCE W. SCHAFER
University of California
Santa Barbara, California

ABSTRACT

This paper describes the ELF operating system structure and discusses a number of considerations which influenced its design. Several applications supported by the system are presented in relation to that structure. The software tools used for constructing and maintaining the system are described, and several implementation problems which were encountered are presented.

INTRODUCTION

ELF is a multiprogrammed operating system which provides a set of programming tools for a computer network environment. The system is implemented for the DEC PDP-11 series computers and is being used to support a number of research applications at various sites in the ARPA network.[1,2]

ELF development was started in early 1973 at the Speech Communications Research Laboratory in Santa Barbara, California. An initial version of the system provided multi-user terminal support capability for network access.[3] In early 1974 that system was tested at several network sites and a decision was made to expand the system structure to provide a more general operating system environment. An initial version of the expanded system was operational at the beginning of 1975, and is currently being used at 30 network sites.

Development of the ELF** system was motivated by the need for a flexible network/user interface. The requirements for such an interface included multi-user terminal access to remote interactive systems, peripheral support for transfer of locally-stored files, real-time data acquisition facilities, and a test-bed for research problems related to computer networks. It was required that these functions be implementable with a variety of hardware and software configurations and be maintainable using the communication facilities provided by the network.

Design of the ELF system draws upon a number of techniques used in other operating systems described in the literature. For example, ELF enables sharing of hardware resources (processor, memory, and I/O devices) by providing a multiprogrammed virtual memory environment; ELF implements a tree structure of processes similar to that found in the TENEX,[4] UNIX,[5] and XDS 940[6] systems.

The ELF system has been structured in a hierarchical fashion in order to simplify its specification and testing, and to provide flexibility of system configuration. Real-time constraints imposed on the system force a compartmentalization of critical paths which disable response to interrupts; limits are placed on the maximum duration of these paths.

A robust inter-process communication facility is incorporated in the system in order to facilitate communication with remote programs (processes) in the network. Programs which utilize ARPA network protocols[7,8] to access remote resources (e.g., files) utilize this inter-process communication facility extensively.

A command language interpreter allows the system to accept requests for service from user terminals, and provides functions for controlling individual user programs in the virtual memory environment provided by the system. A library of these user programs provides capabilities for terminal access to remote systems, Input/Output of digitally-sampled data, transfer of files, or the debugging of new system facilities.

This paper describes the ELF operating system structure and discusses a number of considerations

* This work was supported by the Advanced Research Projects Agency, through Contract No. N00014-73-C-0221 at the Speech Communications Research Laboratory, Santa Barbara, California, and through Contract No. F30602-75-C-0320 at Stanford Research Institute.

** The name ELF is German for "eleven," and is somewhat germane to the naming of IMPs in the ARPA network.

which influenced its design. Several applications supported by the system are presented in relation to that structure. The software tools used for constructing and maintaining the system are described, and several implementation problems which were encountered are presented.

SYSTEM ORGANIZATION

At the center of the ELF system exists a set of modules, collectively referred to as the Kernel, which provides a set of primitive functions for outer-level procedures, and performs tasks which allow the system's processor, memory, and I/O devices to be shared among a set of processes. Kernel primitives perform functions such as dynamic creation of processes, process synchronization, allocation of virtual storage, scheduling of I/O requests, and sharing of an interval timer.[9]

The selection of procedures which reside above the Kernel varies somewhat and is determined by the range of applications to be supported by a system. In ELF systems which provide multi-user terminal support, a set of procedures known as the EXEC provides a mechanism for allocating system resources to users. The notion of a "Job" is utilized at this level to provide an encapsulation of the resources associated with a given user: a tree structure of processes, a collection of open files, allocated virtual storage, and so forth. EXEC procedures utilize Kernel facilities to create Jobs, interpret user commands, run user processes, and maintain the system file structure. User processes which are run under the EXEC provide a variety of functions, including terminal access to remote interactive systems on the network. An illustration of the layered system structure is shown in Figure 1.

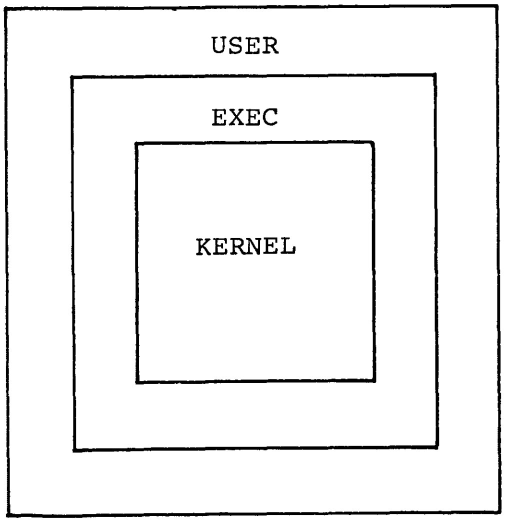


Figure 1—An example of the layered ELF system structure

The Network Control Program (NCP) is another module which may be selected. The NCP uses facilities provided by the Kernel to support a communication mechanism between local and remote processes in the ARPA network. It consists of a set of procedures which establish and control data flow on a set of connections using the ARPA network protocols [Ref].

THE ELF KERNEL

The ELF Kernel concerns itself with three primary areas. The first of these, Processor Management, controls distribution of the processor among a number of asynchronous processes and provides for process synchronization and mutual exclusion. The second major portion of the Kernel is Storage Management, which handles the allocation of physical and virtual storage available to processes in the system. The third portion, I/O management, controls the interaction between processes and external devices, and additionally provides a facility for inter-process communication.

Processor management techniques

The Kernel provides a set of system calls which allow processes to be created, compete for processor service according to priority, inter-communicate, or be terminated. A process is characterized by a virtual program counter, a set of general-purpose registers, a stack, and a queue of elements which are called event messages. Processes are created in the ready state, and remain ready until they are blocked by calling a system primitive for synchronization or mutual exclusion. A scheduling program in the Kernel maintains a list of processes which are in the ready state and transfers control to the highest priority ready process.

Process synchronization

A process synchronization mechanism similar to the message buffer scheme described by Brinch-Hansen is implemented.[10] Each process owns a queue of event messages sent to it by other processes. A process can block itself until a message is added to its event queue by issuing a system primitive called WAIT, which places the process in the "waiting" state if its event message queue is empty. If a process issues the WAIT primitive and elements exist on its event message queue, the process is left in the "ready" state, the event message at the head of its queue is removed and is returned as a parameter.

Event messages are placed on a process' event queue by another process which invokes the SIGNAL primitive, specifying a destination process name and event message. When a process awakens, it receives the 24-bit message in addition to the name of the process which signalled it. In general, the 24-bit message field is interpreted by ELF system processes as an 8-bit "event code" and a 16-bit "data" field. While this assignment of bits is a convention for system processes, higher-level (user) processes which choose to synchronize using SIGNAL and WAIT may use the 24-bit event message field arbitrarily. A system primitive exists to allow a process to wait for a specific event-code.

Examples of synchronization primitives are shown below:

WAIT→⟨PNAME, EVENT, DATA⟩
SIGNAL ⟨PNAME, EVENT, DATA⟩
WAIT-SPECIFIC ⟨EVENT⟩→⟨PNAME, EVENT, DATA⟩

Processor scheduling

Every process that is in the ready state resides in a priority queue that corresponds to a priority level that it was assigned when it was created. The ability of a process to gain control of the processor is a function of the priority queue in which it runs and its position in that queue.

The position of a process within a given queue is determined according to its behavior. A daemon process receives control at regular intervals (currently, every 250 ms.). When it receives control it re-orders the processes within each queue according to their utilization of the processor during the preceding time interval. A process' composite priority is a combination of the priority queue in which it resides and its position in that queue; the process in control of the processor at any point in time is the process at the head of the queue with the highest priority value. This scheme effects a "round-robin" scheduling technique for compute-bound processes while assigning a higher composite priority to processes that are not compute-bound.

Protection mechanisms

Because processes rely on the validity of messages received on their event queues, a protection scheme is required to prevent processes from receiving messages from other non-authorized processes. This mechanism is implemented by means of a capability value which is assigned to a process when the process is created, and may change as the process makes calls to various primitives in the operating system. The access rights of a process executing at a given capability level are a subset of the rights allowed a process with a lower capability value.

A process may be assigned a "branch name" which identifies the processes and all sub-processes in the tree below it. (The assignment of branch-names to processes is utilized by the ELF EXEC in identifying a process as a member of a particular Job.) Processes which have the maximum capability value can SIGNAL any process in the system; processes which are running with lower capability values are limited to the set of processes having identical branch-names.

Mutual exclusion techniques

Processes may request mutual exclusion by means of binary semaphores. Kernel primitives allow dynamic assignment of semaphore names, and provide Dijkstra's P and V operations on those semaphores.[11] Entries may continue to be added to a process' event message queue while the process is waiting for exclusive access to a resource.

Process creation and termination

A process is created by issuing a CREATE-PROCESS system call, specifying a starting address, set of registers, capability value, priority level, and an event code to be used to SIGNAL the creating process when the created process halts itself or encounters a system error (e.g., invalid parameter specification to a Kernel primitive). The system maintains a relationship between each process and its creator, and keeps a list of all processes created by a given process.

A process may be frozen by means of the FREEZE-PROCESS primitive. This causes the process to be blocked and its creator to be signalled with an event message containing the name of the process which was frozen. At this point, the process' registers may be examined or modified, the process may be "thawed," or the process may be terminated. Entries may continue to be added to the event message queue while a process is frozen.

A process may only be terminated by its creator. When a process is terminated, all the processes it has created are also terminated and any resources associated with it (such as outstanding event messages) are released.

Storage management techniques

The Storage Management portion of the Kernel provides a mechanism for creation of a number of virtual address spaces and controls their mapping into physical memory. An address map defines the relation between a user's virtual storage and physical storage addresses. A specific address map becomes associated with a process when the process is created. Any number of address maps may be defined for processes running in user mode; there is a single address map defined in kernel mode, and this is utilized for system (Kernel) primitives. The processor switches from kernel mode to user mode when it gives control to a process; it switches from user mode to kernel mode when a user process makes a system call or is interrupted.

Storage management data structures

There are two data structures used to maintain the relationship between virtual and physical storage. Each virtual address space is mapped as a set of 4096-word pages. A Virtual Storage Map (VSM) is an array which describes the state of each page in a virtual address space. A given page may be undefined (a "hole" in the address space), defined but non-resident, or defined and resident. Each entry in the VSM which describes a resident page contains a pointer to the page in physical storage. A virtual page may also be "read-only," in which case the hardware memory mapping facilities are utilized to prevent modification of shared or protected data. A Virtual Storage Map exists for each virtual address space. Kernel primitives exist for the creation (allocation) of a new virtual address space, and the creation of pages within an address space.

The Physical Storage Table (PST) indicates the relationship between physical pages and the (virtual) pages which occupy them. A mechanism for sharing of virtual pages is provided by the system; in the case of shared pages, the physical storage table points to the head of a linked list of VSM entries. The data structure used to implement this mechanism is shown in Figure 2.

A Virtual Storage Map identifier is included in the system state information for each process. This value uniquely identifies the user address space in which a process is running, and is a logical extension of the process' program counter, as shown in Figure 3. Hardware storage mapping register values are derived from the Virtual Storage Map when the scheduler transfers control to a process.

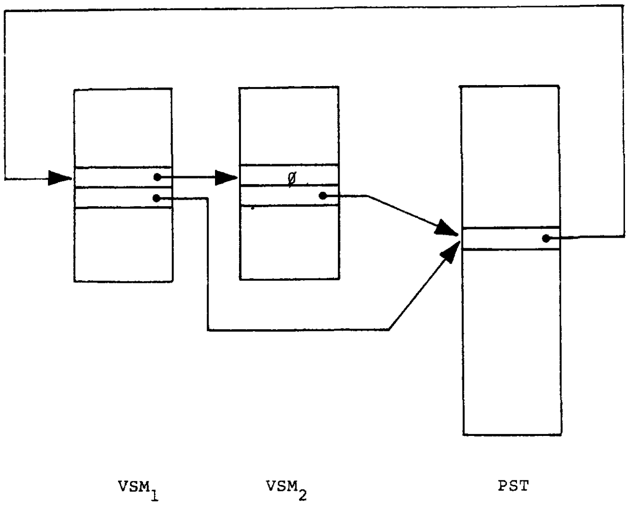


Figure 2—Storage management data structures

Storage management primitives

A set of Storage Management primitives enables a process to allocate a new virtual address space, to allocate pages within an address space, to cause pages to be shared between address spaces, or to block transfer data between address spaces. A new address space is created using the primitive CREATE-VSM, which returns an 8-bit VSM identifier for the new address space. The address space (and all physical storage associated with it) is released when a process issues a DELETE-VSM primitive or when the process which created the address space is terminated.

Creation of a new address space does not allocate any pages within it. A process may define (allocate) new pages within an address space by issuing a DEFINE-PAGE primitive, which allocates a new page in virtual storage and defines the specified page. The caller specifies the Read-Write/Read-Only status of the page. The process may optionally cause the page to be mapped into a page in another address space, causing the pages to be shared between the address spaces. For certain applications (such as system debugging) it is necessary to map a user page into an arbitrary physical page of memory. A privileged (i.e., capability-restricted) primitive allows a process to perform this function.

System primitives require the ability to access parameters which are passed as arguments by a calling process. A system call exists for transfer of a block of data between two address spaces. A primitive may request the identifier of the previous address space in order to obtain or return parameters to its caller.

Hierarchical procedures

ELF allows the establishment of a hierarchical structure of system primitives by means of a run-time binding mechanism. This permits a modular extension of system facilities, such as the addition of file system functions, without requiring modification of the Kernel. The mechanism enables a system initialization process to specify a correspondence between a set of system primitives and the procedures in a user address space which implement them.

When a process makes a call to a system primitive

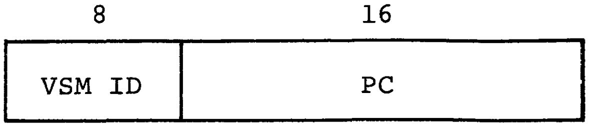


Figure 3—A process' virtual program counter

the system performs some task as an extension of that process. Control may be transferred from the calling process to a primitive in a different address space. In this case the system reloads the hardware mapping registers from the new virtual storage map and transfers control to the primitive in the new address space. The identifier of the previous address space is saved on the Kernel stack to allow restoration of mapping information when the primitive returns.

A process which runs in a user address space may utilize special instructions which "trap" through well-known address vectors on a stand-alone processor; the instructions trap through the corresponding locations in the active user space. A process which is being debugged and reaches a breakpoint, for example, causes control to be transferred to the address contained in the break-point vector (Location 14) of the user address space. This has facilitated the adaption of several debugging packages which run on a stand-alone PDP-11. It is also possible for a process to call a Kernel primitive which requests a signal in the event that a specified process reaches a break-point, and causes the break-pointed process to be placed in the frozen state. A process may thus be responsible for the debugging of a number of processes in the system. The application of this mechanism will be described later.

Input/output management

The Input/Output portion of the Kernel provides a set of system primitives which enable processes to schedule physical I/O requests to devices and to utilize the system's hardware clock. The I/O system queues requests to each device, performs a mapping from virtual to physical storage, provides an inter-process communication facility, and allows the system to be tailored for a particular hardware configuration by selection of a set of device driver modules.

Processes cause the initiation of physical I/O transfers by calling a system primitive which places their request on a queue for a particular device. An I/O process schedules I/O activity on each device, performs storage mapping functions, and causes physical I/O requests to be initiated by calls to device driver procedures.

Input/output primitives

Processes cause requests to be placed on a queue for a particular device by issuing the system primitive START-IO and specifying a device name, a buffer address, a byte count, a function code, and an optional device address. When the requested I/O operation completes, the user process is signalled by the I/O system with the requested event code. Function codes are defined for performing Read, Write, or Device-Specific (e.g., rewind tape) operations. I/O primitives exist for allocation of devices and their optional assignment on a per-process basis. When a sending process issues a write request, it is signalled when all of the bytes it has requested to be written have been transferred. When a process issues a read request, it may specify a data transfer mode (in the I/O function code). A "record" mode read request signals the process (and satisfies its request) when one or more bytes have been read; the reader is informed of the number of bytes actually transferred. A "stream" mode read request signals the reader when the number of bytes requested have been transferred.

INPUT/OUTPUT DEVICE CLASSES

There are three types of I/O devices supported by the system. The first of these classes handles character-oriented devices, such as terminals or line printers, which are used in an interrupt-driven fashion. In the case of character-oriented devices, the system moves the block of data being transferred between the Kernel and the requester's address space. This is necessary to allow interrupt routines to directly address the buffer used for data transfer.

Two utility procedures are made available to device drivers which support terminals. One of these procedures manages a ring buffer to allow type-ahead in the absence of an I/O request from user process. A second utility procedure performs terminal-specific output functions, such as output of padding (ASCII NUL) or expansion of ASCII Horizontal TAB characters. Selection of these two options is made possible by an I/O system primitive for setting device characteristics.

A second class of devices which are supported by the system transfer data on a direct memory access basis and require no processor intervention during the data transfer. The virtual storage structure of the system forces the I/O system to perform a mapping of a user process' I/O buffer into its location in physical storage before calling the device driver which initiates the transfer. (I/O devices which directly access memory do not utilize the processor's memory mapping hardware.) Additionally, the system must determine whether a buffer which is specified in an I/O request crosses page boundaries, and take special action if the associated physical pages are non-contiguous. When this is the case, the I/O system carries out the operation in a piecemeal fashion, initiating separate data transfers for the portions of the buffer residing in each individual page. User processes may avoid this inefficiency by appropriate allocation of buffer space.

A third class of devices provides a mechanism for inter-process communication in a fashion which appears identical to data transfer to a physical I/O device. A set of pseudo-devices, called "Inter-Process Ports" (IPP's) may be used for data transfer between processes. User processes may take advantage of this facility in order to allow flexible assignment of their Input/Output streams; specifically, a process may accept input from either a physically-connected terminal or a remote process on a network. The utilization of Inter-Process Ports for network communication will be described later.

An Inter-Process Port is effectively a mailbox[12] which may be read or written by a pair of processes. A separate read and write request queue is maintained for each Inter-Process Port. Whenever the I/O system receives a matching pair (read, write) of requests, data is transferred from writer to reader. The writer and reader processes are signalled if their respective requests are satisfied. It is possible for the sender to issue an "end-of-stream" indication, which unconditionally satisfies the request of a receiver which is reading in stream mode; the receiver is signalled as usual with the number of bytes actually transferred. Inter-Process Ports may be assigned to a specific pair of processes.

Timer primitives

Timer primitives enable a process to set a number of interval timers and to receive an event message when a timer expires. Additionally, the process may stop a running timer by setting its interval to 0 or may get the current time remaining before a timer expires.

The Kernel maintains a 32-bit Time-of-Day, which is kept in 40-microsecond ticks. Kernel primitives allow user processes to get (or set) the Time-of-Day value. Conversion routines exist outside of the Kernel for obtaining the Time-of-Day as a character string.

THE ELF EXEC

The EXEC is utilized in ELF systems which require a flexible user programming environment. The EXEC interprets user commands and provides a framework for user program support. The command language provides functions such as user identification, display of system status, and initiation of user processes. Design of the ELF command language is patterned after the executive language of the TENEX operating system because of the user-oriented characteristics of that language.[4]

The EXEC support structure provides a set of primitives which are called by user processes below it and includes a terminal control mechanism for interrupting those processes. EXEC primitives allow logical data paths to be established for terminal control and for access to a system file structure; they also perform a number of utility functions, such as data conversion (e.g., time-of-day to string).

The EXEC is implemented as a set of re-entrant procedures which utilize Kernel primitives to support an inverted tree structure of processes. A process called the Logger issues I/O requests to a set of terminals or Inter-Process Ports and "listens" for the arrival of an attention character (control-C). When this occurs, the logger creates an EXEC process which receives characters from the device and interprets user commands. Additionally, it allocates and formats a set of control tables which are used to maintain resources allocated to the newly-created "Job." A Job is identified by a branch-name which is assigned to the EXEC process and becomes associated with all processes in the inverted tree under it. The Job effects a policy for allocation of resources (storage, files, etc.) to each user.

A system primitive exists to enable the logger process to be signalled when a terminal port has been dynamically added to the system. This function allows the system to respond to remote requests for connection from the network and to support a number of virtual terminals. Once a new logger port is established, the logger process issues I/O requests in the same fashion as it would for local terminals; in this case, however, I/O takes place on a pair of Inter-Process Ports.

The EXEC maintains a directory of "sub-systems" which may be dynamically expanded while the system is running. A sub-system is a named collection of procedures which have been loaded into a virtual address space. The EXEC allows the user to initiate a subsystem by typing its name; the EXEC then creates a user process and enables that process to communicate with the controlling terminal by means of EXEC file primitives.

A number of user sub-systems have been written to perform a variety of functions. TELNET, for example, provides the function of terminal access to remote systems on the ARPA network. The TELNET program utilizes file primitives provided by the EXEC to interpret user commands and to establish or terminate connections to remote network Hosts. A cross-network loader, called USERLOADER, allows programs to be loaded into a user virtual address space from a remote file system by means of the file transfer protocol[8] used in the ARPA network.

Generally, a user process is connected to a controlling terminal until the process halts (by freezing itself) or the user interrupts it by typing the EXEC attention character (control-C). I/O to the controlling terminal is re-directed to the EXEC process until the user allows the interrupted sub-system to resume or runs another sub-system. The active tree of user processes is terminated, whenever a user requests to run a sub-system, explicitly issues a "reset" command, or logs out of the system.

Some user processes require the ability to suppress the interrupt facility provided by the EXEC; the user process may specify a "transparent mode" which causes the EXEC to ignore interrupt characters and places this responsibility on the sub-system. This is needed, for example, in the TELNET sub-system, which must be able to transmit the interrupt character to a remote system on the network.

The association between a controlling terminal and a Job may be dissolved by means of a user command or as a result of an error received on a terminal port. The state of the Job and its user processes is saved, and a user may re-attach to it by an EXEC command.

Design of the ELF file system has been aimed at providing a mechanism for terminal control over a set of user processes and the support stream-oriented Input/Output for a variety of device classes. A table-driven mechanism is used to implement stream Input/Output for sequential and file-structured devices. The file structure is compatible with the FILES-11 structure utilized by the DEC RSX-11 operating systems.[13]

The file system provides a primitive for initializing a file path; the file system returns with a Job-unique identifier called a Job File Handle. This identifier may then be used to efficiently call primitives which perform file I/O. The file system supports I/O primitives which read or write relative blocks within a random-access file, and provide a device-independent mechanism for reading or writing a stream of bytes. Stream-oriented primitives exist for byte-input, byte output, string-input, and string-output.

NETWORK COMMUNICATION CONTROL

The Network Control Program (NCP) is an essential component in the ELF system which makes possible process-to-process communication in a network environment. The NCP provides a mechanism for establishing and breaking connections between ELF processes and processes distributed on the ARPA network.

The NCP includes a set of processes which receive messages from the network, interpret messages according to ARPA network standard protocols, format outgoing messages, and control the flow of data on connections by means of the Inter-Process Port facility provided by the ELF Kernel.

A set of network control primitives enables processes within ELF to request the establishment or termination of network connections. When a process calls an NCP primitive to open a connection, the primitive returns the name of an Inter-Process Port which is to be used for data transfer on the connection. When the connection is opened, the process may cause data transfers by calling the normal Kernel I/O primitives using the Port-name which was returned. A process may request that an Inter-Process Port supporting a connection be specifically assigned to the NCP and user processes in order to prevent malevolent processes in the system from performing unauthorized data transfers on the Port.

An important Kernel function which is used by the NCP in the control of data flow on a network connection is that of obtaining the number of bytes requested to be read or written on an Inter-Process Port. This function is necessary to enable the NCP to intelligently determine the amount of data which may be accepted from a remote process. ARPANET protocols utilize a flow control mechanism whereby a sending process must be informed of the buffering capability at the destination.

In addition to control of communication using standard ARPA network protocols, the NCP provides a simple dispatch function for development of experimental protocols. Processes which utilize this function must specify a field which uniquely identifies received messages. Messages are delivered to the user processes (via an Inter-Process Port) in the form in which they arrive from the network.

SYSTEM APPLICATIONS

The ELF system is being used in a variety of applications which require a set of operating system components for network communication. The ELF Kernel provides a base for a variety of software configurations which are tailored according to system requirements. The configuration of processes shown in Figure 4 is typical of ELF systems which provide terminal access to remote systems on the ARPA network. The Kernel, EXEC, and NCP modules support a number of users who access the network by running the TELNET sub-system of the EXEC.

Additional processes may be added to the system for peripheral support. For example, a simple process which uses the ELF NCP to await a remote request for connection may be included in the system. The process accepts a connection, receives a stream of data, and outputs it to a local line printer. A status indication from the NCP signals that a remote process has closed the connection, and the peripheral control process then returns to the "listening" state.

This process provides a simple mechanism for sharing of a peripheral device (in this case, a line printer) among a number of service sites in the network. A daemon process at each service site checks periodically for files to be listed. When they exist, it opens a connection and transmits each file. The ELF NCP queues remote requests for connection while the peripheral process is busy.

A more flexible mechanism for peripheral support is provided by the ARPA network file transfer protocol server process which may be included in the system. The file transfer server responds to remote requests and utilizes facilities provided by the ELF EXEC (file system) to carry out data transfer to a variety of peripherals.

The support of real-time data acquisition functions for speech research has been one of the goals in design of the ELF Kernel. A user process runs as an EXEC sub-system and performs functions of real-time data sampling and file I/O for digitization of speech wave-

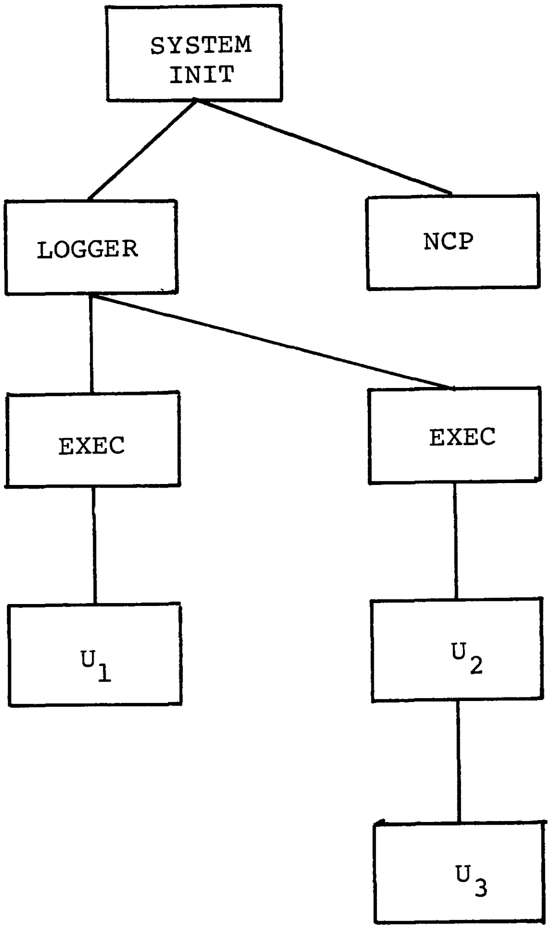


Figure 4—Example process tree

forms. The system may thus be used as a data acquisition station while simultaneously providing a terminal support function for access to remote systems. The sampled signal may be transferred to the remote system by the file transfer protocol process.

The ELF Kernel provides a support base for development of experimental networks and protocols. The packet radio network[14] for example, utilizes the environment provided by the Kernel to support a set of user processes which perform network routing control functions. Additionally, new protocols being tested in the ARPA network utilize the inter-process communication facility provided by the Kernel in the implementation of new network control processes.

DEVELOPMENT AND MAINTENANCE TOOLS

Development and maintenance of the ELF system has made extensive use of software tools provided within the framework of the ARPA network. The ΓENEX operating systems available on the network, for example, have provided a number of services for software development. ELF modules are edited, compiled, and linked together using a set of online subsystems in TENEX. When an ELF system is generated it may be bootstrapped into a target PDP-11 system by transfer of the binary file through the network.

The ELF Kernel, EXEC, and NCP are written in MACRO-11 assembly language. The virtual memory structure provided by the system allows user processes to be written in higher-level languages. Compilers for the BCPL and L1011 Languages are available under TENEX and generate code for the PDP-11. The files produced by these compilers may be transferred to an ELF file structure by means of the file transfer protocol or may be loaded by the ELF USERLOADER sub-system, which was described above.

A number of debugging tools have been developed. A low-level (stand-alone) debugger, called FLEA, was implemented to facilitate checkout of the virtual memory system. FLEA provides the capability of examining or breakpointing any location in physical memory.

Several debuggers which run on a stand-alone PDP-11 have been modified to run in an ELF user address space. Modification was required to make use of ELF I/O primitives for terminal control. Hardware I/O device registers are not available to processes in a user address space.

An additional debugging mechanism involves the interpretation of debug commands received from the network by a system debugging process. This approach minimizes space requirement in a processor being debugged while taking advantage of facilities available on large systems to provide a comfortable user language and a symbolic representation of addresses and instructions. A system debug process* which resides in ELF utilizes the breakpoint signal facility provided by the Kernel and is responsible for debugging of a number of other processes.

ELF system software is distributed as a set of source modules which are accessible in a directory at one of the network TENEX service sites. Users may access the source files by means of the network file transfer protocol. Release-notes and bug-reports (user-feedback) take place through the use of the network message system.

CONCLUSION

The foregoing discussion has presented a structural view of the ELF system as a mechanism for user access to remote resources. A number of design decisions have been made in supporting a range of user applications. At this point we critically examine several of

* Developed at Bolt, Beranek, and Newman, Inc., Cambridge, MA.

these decisions and discuss their relation to the resulting system structure.

The event message scheme which is utilized for process synchronization has proven to be a flexible mechanism which allows processes to wait for multiple events. This capability is required for reliable communication in a network environment: error conditions may arise and "time-outs" are sometimes needed in order to resynchronize a connection sequence. The synchronization mechanism used here is an alternative to the creation of a process for each event; that solution was considered too costly in terms of processor overhead and system table space requirements. Rather, the event message mechanism enables a single process to be multiplexed for several events. Two drawbacks were encountered with the ELF event mechanism, however. First, it is required that any procedure which performs an operation which specifies an event code (e.g., initiation of an I/O request) must receive as a parameter an event code which it may use. In the absence of a statically-allocated set of event codes, an arbitrary system procedure has no knowledge of the set of event codes currently being used by a calling process. A second drawback is the lack of a reliable mechanism for bounding the number of event messages which may be allocated by a signalling process. The strategy of blocking a signalling process when the system's supply of event messages has run out is susceptible to deadlock. For example, the signalling process may be executing an interrupt routine which signals completion of an I/O operation. However, if the process is itself the process being signalled and the supply of event messages runs out the process will deadlock. As a result of these problems, the current system implementation simply returns an error condition to the signalling process, indicating the lack of event message queue space in the system.

The Inter-Process Port mechanism which has been implemented in the system has been valuable in implementing network protocol processes and has provided an effective mechanism for performing local and network I/O in a transparent fashion. There are two disadvantages to that mechanism. First, a Port mechanism requires buffer space in each address space which communicates using the port. Second, processing time is consumed in the copying of data from the sender to the receiver's buffer space. It is believed that these disadvantages are outweighed, however, by the system flexibility introduced by a general inter-process communication mechanism.

The memory management scheme used in ELF has been designed to provide a simple mechanism for allocation of physical storage and to provide a means for sharing of user pages. A problem with the current storage allocation scheme is that of internal fragmentation which results when a number of small user procedures reside in a virtual address space or when a large number of user processes wish to share a small amount of data. A modification of the storage management portion of the Kernel to provide for variable-sized segments is currently under consideration.

The choice of a hierarchical structure of system functions has provided modularity and has facilitated checkout of the system. Its drawback is an occasional overhead introduced in the passing of parameters between levels. Provision of standard memory management functions which facilitate these tasks in hardware would alleviate this problem.

SUMMARY

This paper has presented a description of the ELF Operating system which provides a set of user access facilities for the ARPA computer network. We have attempted to describe the system from a structural point of view and point out operating system functions which are necessary in a network environment.

ACKNOWLEDGMENTS

In addition to the work done by the authors, significant contributions to the design and implementation of the system were made by Mr. Jon Miller, who was responsible for the I/O and Storage Management portions of the Kernel, and Mr. Jim McClurg, who participated in implementation of the file system and maintained system documentation. Contributions were also made by V. Strazisar and E. Mader at Bolt, Beranek, and Newman, Inc., and by Mr. P. Raveling of University of Southern California Information Sciences Institute.

REFERENCES

1. Roberts, L. G., B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," *Proceedings of AFIPS SJCC* 1970, pp. 543-549.
2. Heart, F. E., R. E. Kahn, S. M. Ornstein, W. R. Crowther, D. C. Walden, "The Interface Message Processor for the ARPA Computer Network," *Proceedings of AFIPS SJCC* 1970, pp. 551-567.
3. Retz, D. L., "ELF—A System for Network Access," *Proceedings of IEEE Intercon*, New York City, April, 1975.
4. Bobrow, D. G., J. D. Burchfiel, D. L. Murphy, R. S. Tomlinson, "TENEX, a Paged Timesharing System for the PDP-10," *Communications of the ACM*, Vol. 15, No. 3, March, 1972, pp. 135-143.
5. Richie, D. M., K. Thompson, "The UNIX Time-Sharing System," *Communications of the ACM*, Vol. 17, No. 7, July, 1974, pp. 365-375.
6. Lampson," B., et al., "A User Machine in a Time-Sharing System," *Proceedings IEEE 54*, 12 December, 1966, pp. 1766-1774.
7. McKenzie, A. A., *HOST/HOST Protocol for the ARPA Network*, National Technical Information Service, AD757680.
8. Crocker, S. D., R. M. Metcalfe, J. B. Postel and J. F. Heafnet, "Function-Oriented Protocols for the ARPA Computer Network," *Proceedings of AFIPS SJCC* 1972, Vol. 40, pp. 271-279.
9. Retz, D., J. Miller, J. McClurg, B. Schafer, *ELF System*

Programmer's Guide," Speech Communications Research Laboratory, Santa Barbara, Calif., September, 1974.

10. Brinch-Hansen, P., *Operating System Principles,* Prentice-Hall, July, 1973.
11. Dijkstra, E. W., The Structure of the "THE"-Multiprogramming System," *Communications of the ACM,* Vol. 11, No. 5, May, 1968, pp. 341-346.
12. Spier, M., E. Organick, "The MULTICS Interprocess Communication Facility," *Proceedings of the ACM Second Symposium on Operating System Principles,* Princeton University, October 20-22, 1969, pp. 83-91.
13. Digital Equipment Corporation, *RSX-11 I/O Operations Manual,* Order No. DEC-11-OMFSA-B-D.
14. Burchfiel, J., R. Tomlinson, M. Beeler, "Functions and Structure of a Packet Radio Station," *Proceedings of AFIPS NCC* 1975, Vol. 44, pp. 245-251.

Elements of a planning and modeling system

by THOMAS H. NAYLOR
Duke University and Social Systems, Inc.
Chapel Hill, North Carolina

ABSTRACT

Today there are nearly two thousand corporations in North America and Europe either using, developing, or experimenting with some form of corporate planning model. With the emergence of this new and rather substantial interest in the methodology of corporate planning modeling, there appears to be a definite need for a conceptual framework which can be used to design and implement computer based planning and modeling systems.

In this paper we describe a collection of elements which we believe to be of critical importance in designing a corporate planning model. Our objective is to develop a set of criteria for not only designing a planning and modeling system, but a set of criteria which can also be used to facilitate the evaluation and comparison of alternative planning and modeling systems.

There are nearly 50 planning and modeling software packages on the market today. These include systems such as BUDPLAN, COMOS, and SIMPLAN. This paper attempts to provide the reader with a convenient check-list of possible features to consider in either designing one's own system or selecting an appropriate software package.

INTRODUCTION

A recent survey by Naylor and Schauland confirms the fact that nearly 2,000 corporations in North America and Europe are either using, developing, or experimenting with some form of corporate planning model. For the most part, these models are "what if" simulation models capable of generating alternative futures and scenarios depending on the policy assumptions and external assumptions made by corporate management. Our survey indicated that less than 4 percent of the models in a sample of 346 companies were optimization models. The rest were simulation models.

We believe there are eight basic elements which one must consider in designing a planning and modeling system:

1. Planning System
2. Management Information System
3. Modeling System
4. Forecasting System
5. Econometric Modeling
6. User Orientation of the System
7. System Availability
8. Software System.

PLANNING SYSTEM

The point of departure for any corporate planning model is the planning system itself. That is, the design of the planning system for the organization should be set in place before any consideration is given to the modeling system.

As the most general case let us consider a large, decentralized company consisting of multiple divisions, groups, products, or strategic business units. For simplicity, we shall refer to any such sub-system of an integrated company as a *business unit*. Each business unit is assumed to be autonomous and is responsible for its own marketing and production activities. Although cash management and overall corporate financial planning are centralized at the corporate level, each business unit is responsible for its own income statement.

At the beginning of the planning cycle, global goals and objectives for the company are specified by top management and interpreted to the business units by the corporate planning department. These corporate goals may take the form of specific target objectives for the company as a whole or for individual business units. Typical target variables may include return on investment (ROI), market share, sales growth, and cash flow, as well as environmental, social, and political objectives.

The corporate planning department designs the report formats to be employed by the business units in formulating their business plans. Standardized reporting at the business unit level greatly facilitates the consolidation of plans across all business units. However, individual business units are permitted to make their own assumptions concerning marketing and production provided they are explicit about the external assumptions and policy assumptions underlying their business plans. Financial plans at the business unit level follow logically from given assumptions about revenues and costs.

Plans from the business units are transmitted to the corporate planning department for consolidation, review, and evaluation. In the initial stages of the planning process, the business plans will be returned to the business units for modification and reformulation in light of corporate goals. This iterative process will be replicated until all of the business plans have been approved and consolidated into the company's overall corporate plan.

In the following section we shall describe how business planning models can be integrated into the planning process. The integration of planning models into the planning process is perhaps the single most difficult step in the entire process of corporate modeling. Relatively few companies have successfully integrated corporate planning modeling into the planning process. Two notable exceptions are the Wells Fargo Bank and the Central National Bank of Cleveland.

Business planning models

Figure 1 contains a flow chart of a consolidated corporate planning model which is driven by a series of business planning models for the individual businesses of the company. These models may either be used on a stand alone basis at the business unit level or consolidated and used by the corporate planning department, senior financial officers, or the chief executive officer. Each business unit model consists of a front-end financial model driven by a marketing model and a production model.

The objectives of the business unit models are to generate alternative scenarios and business plans based on varying assumptions about business unit policies and assumptions about the external environment of the businesses.

Financial planning models

Each business planning model produces as output data a proforma income statement for the business unit. In cases where the business unit is actually a subsidiary of the parent company, then proforma balance sheets and sources-and-uses of funds statements may be produced as well. Basically, these business financial models can be used to simulate the effects on net profit of alternative business strategies for a given business unit. The validity of the results generated by a business unit financial model will be no better or worse than the assumptions underlying the revenue and production cost projections which feed the model.

Marketing planning models

Marketing planning models provide the revenue projections which drive the business planning models. Two alternatives are available—forecasting models and econometric marketing models. Forecasting models

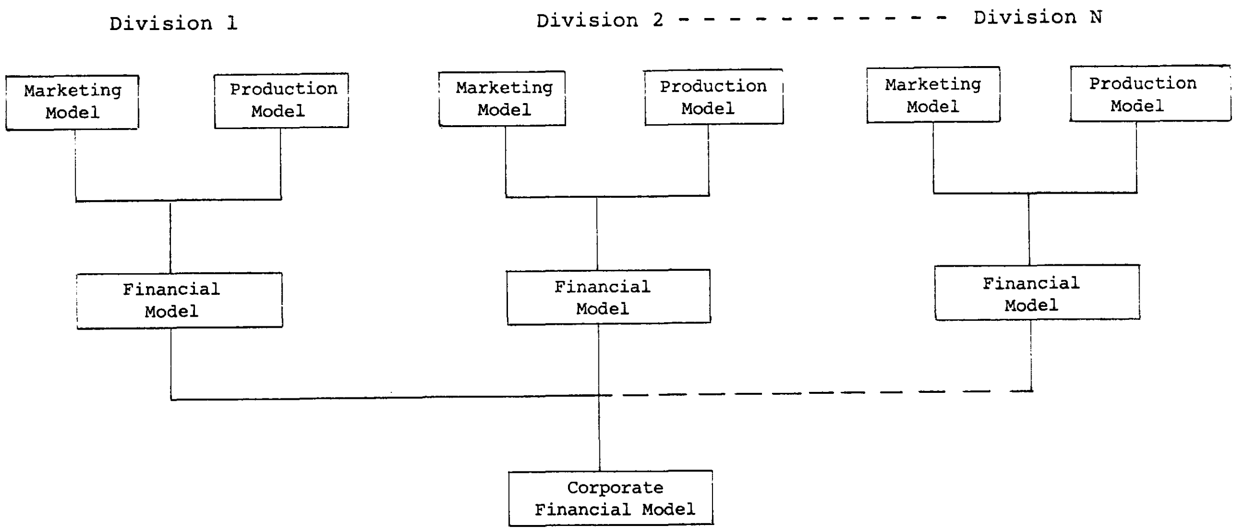


Figure 1—A Conceptual Framework for Corporate Models

are naive, mechanistic, time-series techniques which attempt to forecast next month's sales in terms of last month's sales, sales the month before that, and the month before that. Forecasting models are void of explanatory power and cannot be used to simulate the effects of alternative marketing strategies or alternative assumptions about the national economy on sales or market share. Econometric models, on the other hand, can be used to improve one's understanding of the "market" and to simulate the effect on sales volume, sales revenue, and market share of alternative pricing, advertising, and competitive strategies as well as alternative assumptions about the national or regional economy.

Production planning models

Given a sales forecast for a particular business unit, how much will it cost to produce at a level which will satisfy the demand forecast? That is the *raison d'etre* for production planning models. A number of companies including Monsanto and Inland Steel use a type of activity analysis approach to production planning modeling which generates the cost-of-goods sold associated with a given demand forecast. A reasonable extension of this approach is for the production model to generate the minimum cost production plan associated with a given level of demand for the products of the business unit. This latter alternative represents a logical interface between mathematical programming and other optimization techniques and corporate simulation models.

Consolidated corporate planning models

As we have previously indicated, the individual business planning models may either be used as planning tools for the separate business units, or consolidated at the corporate level to form consolidated corporate plans. The corporate planning department should have the option to perform "what if" experiments with any of the business unit models on either a stand alone basis or as part of a totally integrated planning and modeling system.

The output reports of a consolidated corporate planning model typically include proforma income statements, balance sheets, and sources-and-uses of funds statements. Our survey indicated that of those firms which have some type of corporate planning model the following applications are the most prevalent: (1) cash flow analysis, (2) financial forecasting, (3) profit planning, (4) budgeting, (5) investment analysis, and (6) merger-acquisition analysis.

In summary, a corporate planning and modeling system should have the ability to integrate finance, marketing, and production at the business unit level and the ability to run the business unit models separately or as part of a consolidated corporate financial planning model. Whether the company engages in "top down" or "bottom up" planning is less important than whether or not the planning modeling system can be easily adapted to the planning system. Regardless of the type of planning system, two features are critical—(1) the ease with which financial, marketing, and production models can be integrated and (2) the ease with which financial consolidation can be achieved.

MANAGEMENT INFORMATION SYSTEM

Given a planning system, the next important element in the development of a corporate planning modeling system is the management information system. We shall define the term management information system to include the following elements: (1) database, (2) database management system, (3) security system, (4) report generator, and (5) graphics.

Database

A decision to develop a corporate planning model is tantamount to a serious commitment to the maintenance of comprehensive internal and external databases. Our database requirements become quite explicit once we settle on the design for a corporate planning model.

Internal data

To develop an annual financial planning model we need at least three or four years of historical financial data. We require even more data if the model is a monthly or quarterly model. Econometric marketing models should have 25 to 30 observations of historical data. While most firms have little or no difficulty meeting the data requirements for financial models, data problems are much more severe in the case of marketing and production models.

External data

Most econometric marketing models attempt to link sales volumes and sales revenues to the national or regional economy in which a particular product is sold. A number of service bureaus offer national historical macroeconomic data and econometric forecasts to their clients. These services tend to be quite expensive and the econometric forecasts offered by the bureaus have not been noted for their accuracy in recent years.

An inexpensive alternative to the use of an econometric forecasting service is to subscribe to the historical database of the National Bureau of Economic Research (NBER). The fee for the NBER database is quite nominal and includes over 2,200 economic time-series. In other words, use the NBER data base to specify, estimate, and validate your econometric models. Then either make your own assumptions about the economy of the United States or subscribe only to the quarterly forecasts of one of the aforementioned econometric service bureaus. It is possible to purchase the forecasts for considerably less than a "full service" contract from one of these bureaus. The NBER database is available through most timesharing bureaus and can also be installed on the user's in-house computer.

Database management system

Not only should a planning and modeling system have databases, but it should also have a flexible, easy-to-use database management system for reading data into the system, storing it, and making it readily available for modeling and report generator. At last three different approaches to database management have emerged among the numerous planning and modeling software systems which are currently available—(1) matrix, (2) row-column, and (3) record-file.

Matrix

PSG II, a FORTRAN based planning system as well as several APL based planning and modeling systems use matrices to read data into the system. Both database management and modeling functions are carried out using matrix manipulations. If the user is a scientific programmer, matrix manipulations should cause no problems. However, many corporate planners and financial analysts are neither mathematicians nor scientific programmers and may find matrix manipulations difficult, if not impossible to use.

Row-column

Other planning and modeling systems such as FORESIGHT make use of row numbers and column numbers to create databases, formulate models, and generate reports. While the row-column number approach may have some appeal to accountants who are accustomed to working with financial spread sheets, the user must keep track of the row and column numbers. Furthermore, econometric and production data do not necessarily lend themselves to this restricted notation.

Record-file

Other planning and modeling systems like SIMPLAN make use of records as the basic unit of data. A *record* is a time series variable such as SALES, COST, or PROFIT. A record has a name, an abbreviation, a value, units, and a security level which determines who has access to which SIMPLAN records. A *file* is a collection of SIMPLAN records. Each business unit may have one or more SIMPLAN files. For example, for a given business one file may contain actual historical data. Another file may contain budgeted, projected, or simulated series. Variance reports and validation runs are particularly easy to implement with the multiple file concept.

Security system

There should also be some means of controlling who has access to which files, records, models, and reports within the planning and modeling system. Division managers should be able to access their own databases, models, and reports, but not those of other divisions or the entire corporation. Corporate management should, on the other hand, be able to access the corporate database as well as all division databases, models, and reports. A built-in security system makes all of this possible.

Report generator

The front-end of any planning system is a set of financial reports. Therefore, it follows that a report generator should be an integral part of any planning and modeling system. Basically, management should be able to have any type of report format it desires. That is, the report generator should not impose any restrictions on the type of report which is produced by the system.

The report generator should be flexible and easy-to-use. Some report generators are so easy-to-use that typists with no previous programming experience can be taught to produce financial reports with little or no effort.

Of the 50 planning and modeling software packages available today, over two-thirds of them are primarily report generators. That is, they can produce financial reports and do financial consolidations, but have very limited database management, modeling, and econometric features. PROPHIT II, FAL, FORESIGHT, and INFOTAB are examples of financial report generators. Although financial report generation and financial consolidations are important elements in a planning and modeling system, there are other important elements to consider. Unfortunately, a number of users of systems which are primarily report generators have found themselves locked into expensive outside timesharing charges only to realize, when it is too late, that they need additional database management, modeling, econometric, and forecasting features which are not available in their financial report generator. Although simple financial modeling and report generation are ideal starting points for those who are just beginning to develop a planning and modeling system, beware of dead-end systems which can only do report generation and are not available for installation on your in-house computer.

In summary, in selecting a report generator, make sure that the planning and modeling system of which it is a part has the flexibility and features which you will need in the future as well as the present.

Graphics

An increasing number of planning and modeling software packages now offer graphics as an alternative way of displaying output data and corporate plans. The graphical display of sales, cost, and profit trends can be an effective way to present planning data to top management.

MODELING SYSTEM

Given the discrete nature of business planning data, virtually all corporate planning models take the form of finite difference equations. In this section we shall describe five modeling features which the user may want to include in a planning modeling system: (1) recursive models, (2) simultaneous models, (3) logical models, (4) risk analysis, and (5) optimization.

Recursive modeling

Most of the financial planning models which have been developed to date are recursive or causally ordered models. That is by placing the equations of the model in the proper order it is possible to solve each equation one at a time by substituting the solution values of previous equations into the right-hand side of each equation. Recursive models have the computational advantage that you do not have to resort to matrix inversion or some other simultaneous equation technique to solve the system of equations. Below is an example of a recursive financial model.

```
1  SALES    =A-B*PRICE
2  REVENUE  =PRICE * SALES
3  CGS      =.60 * REVENUE
4  PBT      =REVENUE-CGS
5  TAX      =.50 * PBT
6  NPR      =PBT-TAX
```

In this example, the selling PRICE is given. A and B are parameters and

SALES	=Sales volume (units)
REVENUE	=Sales revenue
CGS	=Cost of goods sold
PBT	=Profit before taxes
TAX	=Taxes
NPR	=Net profit

But in many financial models it is impossible to express the logic of the model as a series of recursive, causally ordered equations. It is for this reason that a corporate modeling system should include the capability to solve simultaneous equation models as well as recursive models.

Simultaneous models

Consider the following five-equation financial model.

```
1  INT     =.12 * Debt
2  PROFIT  =REVENUE-CGS-INT-TAX
3  DEBT    =DEBT(-1)+NDEBT
4  CASH    =CASH(-1)+PROFIT+NDEBT
5  NDEBT   =MBAL-CASH
```

Profit (PROFIT) in Equation 2 is defined as sales revenue (REVENUE) less cost of goods sold (CGS), interest (INT), and taxes (TAX). But INT depends on total indebtedness (DEBT) in Equation 1. From the balance sheet total debt in Equation 3 for this period is equal to last period's debt DEBT(−1) plus new debt (NDEBT). New debt is defined in Equation 5 as the difference between the cash balance (CASH) and the firm's minimum required cash balance (MBAL). CASH in Equation 4 is the sum of last period's CASH, PROFIT, and NDEBT.

Although this model is quite simple it is, nevertheless, a simultaneous equation model. It is impossible to solve the model recursively merely by placing the equations in the correct order. Solution of this model requires the use of a technique capable of solving simultaneous equations. This model may either be solved through matrix inversion techniques or some other generalized technique such as the Gauss-Seidel method which is suitable for both linear and nonlinear simultaneous equation models.

As you can see from our example model, it is indeed quite likely that we will encounter simultaneity even in quite simple financial models. Very few of the financial modeling software pakages have the ability to solve simultaneous systems of equations. CUFFS, SIMPLAN, and XSIM are exceptions to this rule. SIMPLAN and XSIM can handle linear and nonlinear simultaneous equation models. CUFFS can solve linear models.

Simultaneous equation problems can also arise in econometric marketing models where two or more products are either complements or substitutes.

Many banks have developed a special type of financial planning model known as an asset-liability model. Most of these models have been formulated as recursive models. Yet logically this is totally absurd, for the very nature of a bank's assets and liabilities is a simultaneous jointly determined structure. For example, demand deposits and time deposits are substitutes and both are likely to be correlated with various loan demand equations. Loan demand depends on interest rates and interest rates depend on the supply and demand for loans. Consumer loans and mortgage loans may be substitutes for one another. To attempt to model the asset-liability structure of a bank with a recursive model makes little or no sense. It is not surprising to find that a number of banks have encountered serious difficulties in attempting to validate recursive, bank planning models. The financial structure of a bank is simply not recursive.

Logical models

The ability to check whether or not cash balances or inventory levels have dropped below some predetermined minimum level is another important element of a planning modeling system. Logical commands such as an IF statement or a GO TO command are desirable features for a planning modeling system.

Risk analysis

A strong case can be made for treating some of the external variables in a corporate planning model as random or probabilistic variables with given hypothetical or empirical probability distributions. This type of analysis is known as *risk analysis.* Risk analysis is useful in testing the sensitivity of the planning model to random shocks and perturbations, constructing confidence intervals, and testing hypotheses. But our survey of 346 corporations indicates that of those firms which have some form of corporate planning model, only 6 percent make use of risk analysis.

There are two major reasons why risk analysis has been used so seldom with planning models. First, use of risk analysis with corporate planning models is prohibitively expensive. Plan on multiplying your computer bill for the deterministic version of your model by a factor of 100 if you use risk analysis. Second, risk analysis is difficult to explain and interpret to management.

Some analysts use a type of pseudo-risk analysis in which they experiment with "optimistic," "pessimistic," "most likely" values of external variables rather than treating them as random variables.

Optimization

As we have previously indicated, only 4 percent of the users of corporate planning models identified in our survey were found to be using their models as optimization models. Those firms which do not use optimization techniques in conjunction with corporate planning models tend to use them for production planning rather than as global optimization models for an entire business or the corporation as a whole.

Although optimization models are widely used in certain process industries such as oil refineries, rarely are these production scheduling models integrated into a corporate planning model. Virtually every major oil refinery in the world uses mathematical programming to schedule its operations. At this point in time we are not aware of a single oil company which has a linear programming model linked to a corporate planning model.

The difficulty with using optimization techniques to develop optimal plans for a corporation as a whole is a problem of problem definition. Although top management is indeed interested in profits, ROI, discounted cash flow or some equivalent measure of performance, these are by no means the only measures of effectiveness which management uses to evaluate corporate plans. Output of corporate planning models is a vector not a single variable. If the company wants to survive, management must necessarily monitor a whole host of output variables—profit, ROI, market share, sales growth, cash flow, as well as all of the line items of the income statement and balance sheet.

Faced with a multiple output planning problem, optimization techniques which optimize with respect to a single output variable are of limited use to corporate planners. The use of *goal programming* and *utility theory* have been suggested as means of quantifying trade-offs among conflicting corporate objectives. The track records of these two techniques as corporate planning tools are not impressive.

Although the energy crisis, shortages of a variety of production inputs, and inflation may cause more firms to utilize optimization techniques for production planning modeling, we do not foresee significant usage of these techniques as global optimization techniques for overall corporate planning. However, we do expect them to be used more often as production planning tools at the business unit level and integrated into business financial models.

Very few of the existing planning and modeling languages have mathematical programming routines incorporated into their structures. COMOS, the planning and modeling system developed by CIBA-GEIGY, does have this feature. Some planning and modeling systems have the ability to interface and exchange files with mathematical programming packages.

FORECASTING SYSTEM

The ability to generate short term forecasts not only for market planning models but for any external variable which appears to have a reasonably stable relationship with respect to time is another important element to be considered for inclusion in a planning modeling system. A variety of short term, "naive," forecasting tools are available ranging from simple time trends to the Box-Jenkins technique in terms of degree of complexity. Although short-term forecasting models have a definite role to play in corporate modeling, they have little or no explanatory power and cannot be used for "what if" analysis.

Time trends

Probably the most straightforward forecasting models are simple linear, quadratic, exponential, or logarithmic time trends which express sales volume, for example, as a function of time only. The parameters are estimated by ordinary least squares techniques.

Exponential smoothing

Exponential smoothing techniques consist of a set of weighting schemes which assign greater weight to more recent historical observations than those from the more distant past. Again the rationale is the same. To forecast the future all one needs to know is the correct relationship between past sales and future sales. The problem of exponential smoothing is one of selecting the approximate weighting scheme.

Adaptive forecasting

Adaptive forecasting models are a collection of techniques which have the ability to "self correct" if the forecast is not tracking the actual behavior of the system. Adaptive forecasting techniques are much easier to use than Box-Jenkins techniques and have been known to perform equally well.

Box-Jenkins

Box-Jenkins techniques are the most powerful, most sophisticated, and most difficult to use forecasting techniques available. They are not techniques for amateurs. In fact, the user will probably need a mathematical statistician to hold his hand while using these complex procedures. Through a set of "transfer functions" it is possible to link Box-Jenkins forecasts to a set of external leading indicators.

ECONOMETRIC MODELING SYSTEM

If the user wants to do computer simulation experiments simulating the effects on sales volume or market share of alternative pricing, advertising, and competitive strategies, then econometric models are the appropriate analytical tools. Econometric models can also be used to link market forecasts to the national and regional economies. Finally, our understanding of the market behavior of specific products or groups of products can be considerably enhanced through the use of econometric marketing models. But the forecasting accuracy of any econometric marketing model is no better than the accuracy of the policy assumptions and assumptions about the firm's external environment which underlie the model.

Methodology

Econometric modeling involves a four-step methodology which will be summarized below. These steps include: (1) model specification, (2) parameter estimation, (3) validation, and (4) policy simulation. Given the present state of development of computer software, it is now possible to implement all four of these steps within the planning modeling system without having to go out of the system to FORTRAN, PL/1, or some other type of subroutines. SIMPLAN and XSIM are among the very few modeling systems which have a fully integrated econometric modeling capability.

Specification

Unfortunately, most econometrics textbooks are concerned only with the question of "Given an econometric model, how do we estimate the parameters of the model?" In other words, the entire question of model specification has been assumed away by most textbooks and university courses on econometrics.

The specification of econometric marketing models requires: (1) considerable knowledge of the market of the product or group of products being modeled, (2) familiarity with econometric and statistical methods, and (3) some knowledge of microeconomics and the theory of markets.

If multiple product econometric models are to be developed, we recommend the use of a well designed questionnaire to be used by analysts in extracting relevant market information from product managers. Such a questionnaire can greatly reduce the amount of interaction time between analysts and product managers.

Estimation

Single-equation econometric models can be estimated using ordinary least-squares (OLS) regression techniques. Simultaneous-equation models require the use of techniques like two-stage least-squares (TSLS) or other simultaneous-equation estimators. The application of OLS to simultaneous-equation models may yield biased, inconsistent estimates. Most of the planning and modeling software packages include OLS, but very few of them offer TSLS or other simultaneous equation estimators.

Validation

The ultimate test of the validity of an econometric model is how well it forecasts the actual behavior of the system it was designed to emulate. This implies solving the model each period for the output variables in terms of given policy variables and external variables as well as lagged values of the output variables generated by the model in preceding time periods. In other words, the model is viewed as a closed loop dynamic system which is driven by a set of starting values for the lagged output variables and given values for the policy and external variables.

Since econometric models may either be linear or nonlinear and either recursive or simultaneous, some technique like the Gauss-Seidel method is needed to solve the simultaneous equation models. Ideally, simple one-word commands like SOLVE and VALIDATE can be used to solve and validate econometric models. It is also desirable to produce a comparison of simulated and actual values and perhaps compute mean percent absolute errors for each output variable.

Simulation

Finally, once we have specified, estimated, and validated an econometric model which we feel we can live with, we are then ready to conduct policy simulations with the model. We simply change the policy variables and external variables and solve for the output variables. Again we need a technique like the Gauss-Seidel method to solve the simultaneous equations.

Integrated models

Although estimation, validation, and policy simulation are, in fact, three separate computer programs, it is possible to integrate each of these steps into a single system so that the user can move easily from one step to another. Commands like ESTIMATE and TSLS can be used to estimate the parameters with ordinary least-squares and two-stage least-square respectively. In addition, a set of test statistics for each equation will also be produced—R,^{2} t-statistic, F-statistic, standard errors, Durbin-Watson statistics, etc. VALIDATE and SOLVE commands generate the time paths of the output variables for validation purposes and policy simulation.

Some systems also contain a SAVE command which enables the user to save the structural specification and parameter estimates of an econometric marketing model and pass them on to a financial model without ever leaving the system. With this feature, it is quite easy to integrate financial, marketing, and production models. No longer is it necessary to develop econometric models on one system and then re-code them for use on a different system if one wants to use the econometric results for planning. Econometric modeling as well as forecasting modeling can now be fully integrated into the planning modeling system.

National and regional economic models

It is also possible to link national econometric models and economic databases directly to a planning and modeling system. For example, Monsanto and Dresser Industries each have the Wharton Econometric Forecasting Model installed on their in-house computer and linked to their business planning models. Hundreds of firms use modeling systems which are linked to the NBER economic database on several timesharing service bureaus.

USER ORIENTATION OF THE SYSTEM

Up to this point we have described a number of basic elements which we believe to be worthy of serious consideration in the design of a planning and modeling system. Various subsets of each of these elements are available in the form of special purpose computer software packages. For example, RAMIS, NOMAD, TOTAL, and INS are all excellent database management systems. FAL, INFOTAB, PROPHIT II, PSG II, and FORESIGHT are all financial report generators. ESP, TSP, ECON, and SPX are econometric and statistical estimation packages. Many of these software packages are quite well suited for special purpose functional applications.

But if our objective is comprehensive corporate planning and modeling then we are likely to require (1) a database management system, (2) a security system, (3) a report generator, (4) a simulation modeling system, (5) a forecasting system, and (6) an econometric modeling system. And, furthermore, it would be extremely convenient to have all of these features linked together as subsystems of a truly integrated planning and modeling system.

Ease of use

It is one thing to advocate an integrated planning and modeling system consisting of the six subsystems described in the preceding paragraph, but what if the resulting system is an extremely cumbersome, difficult-to-use system which requires the user to be a senior programmer or computer scientist? Fortunately, recent breakthroughs in computer science and corporate modeling techniques have made it possible to design and implement an easy-to-use planning and modeling system which includes all six of the subsystems described in this paper. More will be said concerning the ease of use of planning modeling systems, when we discuss computer software systems.

User specified subroutines

Although we have advocated a planning and modeling system which contains a substantial number of powerful built-in functions and subroutines, we recognize the impossibility of building a system which is all things to all people. There will always be a user who wants some special subroutine to satisfy his own unique needs. With this thought in mind, an integrated planning and modeling system should be sufficiently open ended to permit the user to write his own subroutines in, for example, FORTRAN or PL/1. With this feature, the user never gets locked into a particular system.

SYSTEM AVAILABILITY

Corporate planning modeling systems may either be run interactively or in batch either on the user's in-house computer or on an outside service bureau. Although computer service bureaus, particularly time-sharing bureaus, may provide a convenient vehicle for the development and testing of individual business unit planning models, putting an integrated comprehensive total corporate planning model and database up on an outside service bureau is likely to be prohibitively expensive. The disk charges for the corporate databases alone will be enormous. Over the long run, we believe that most of the really serious corporate planning and modeling systems for large companies will be implemented on in-house computers rather than on an outside bureau. However, smaller firms which are equivalent to single business units in our Figure 1 flow chart, will still find service bureaus to be the most cost effective alternative for doing financial planning and modeling.

Interactive

All things being equal, it is difficult to argue against the merits of interactive computing for corporate planning and modeling. The benefits of conversational computing to planning are obvious and well documented in the literature. But interactive computing can be quite expensive even on in-house computers, if one considers the opportunity cost of alternative uses of computer central processing units. Therefore, we recommend interactive computing during the model debugging stage and when the timeliness of alternative plans and scenarios justifies the premium charges for interactive computing.

Batch

Batch computing is more appropriate for creating large historical databases and doing multi-scenario production runs where the user is not faced with an urgent deadline to make a decision.

SOFTWARE SYSTEM

What about the task of programming a corporate planning and modeling system? Basically, two alternatives are available. The system can either be programmed in a general purpose scientific language like FORTRAN, PL/1, or APL or it can be coded in a planning and modeling language like BUDPLAN, COMOS, or SIMPLAN.

There are at least two major benefits associated with the use of one of the scientific programming languages. First, they are extremely flexible. That is, every feature which we have proposed for a planning and modeling system could be coded in FORTRAN, PL/1, or APL. Indeed, our survey showed that 50 percent of the corporate models in our sample had been written in FORTRAN. Second, these languages are quite well-known, particularly FORTRAN.

But there are some very serious limitations to the use of scientific programming languages for corporate planning models. First, corporate planners and financial analysts may not be familiar with any of these languages since they may not have previous computing experience. Second, database management and report generation are not the main strengths of FORTRAN and APL. (PL/1 admittedly has some features which facilitate file manipulation and report generation.) Third, these languages offer little assistance in either formulating or coding corporate planning models, since they are general purpose scientific languages. Fourth, it is the rule rather than the exception for top management to make frequent changes in their requirements in terms of report formats, policy assumptions, external assumptions, types of consolidations, etc. Mergers and acquisitions occur, new products are introduced, and old products are dropped. These types of changes are not easy to implement with scientific programming languages. A major reason for the demise of most of the large-scale models developed in the 1960's was their lack of flexibility. Without exception, the Sun Oil, Xerox, and *New York Times* models, as well as several others, were all written in FORTRAN. When Sun Oil merged with another oil company, the model was dropped rather than re-programming it in FORTRAN. Fifth, even if the model builders are accomplished programmers, econometric modeling is very difficult with scientific programming languages.

Some have suggested that APL will be the wave of the future for corporate modeling. Although APL is by far the most powerful scientific language available today, it has some unique disadvantages which are likely to render null and void the fantasy of corporate managers sitting at their APL terminals doing corporate planning. First, APL assumes the user is proficient at mathematics including matrix algebra. This assumption simply does not hold up in the real world. Very few managers have ever been exposed to matrix algebra. Second, the special characters and mathematical operators of APL are likely to be foreign to most managers, financial analysts, and corporate planners. In summary, APL is a fantastic language for computer scientists and mathematicians, but its utility as a corporate planning tool is severely limited.

The alternative to scientific programming languages is to use one of the new planning and modeling languages designed specifically to facilitate the formulation and coding of corporate planning models. Among the benefits to be derived from using one of these planning and modeling systems are the following. First, they are easy to use. To do financial modeling with a system like SIMPLAN, the user must be familiar with high school algebra, accounting, and finance. The user need not be familiar with modeling or computer programming. Second, some of these systems provide a conceptual framework for planning and modeling which makes it much easier to develop the model in the first place. Third, with a select few of these systems, it is possible to have all six of the following subsystems integrated within the planning and modeling system: (1) database management, (2) security, (3) report generation, (4) simulation modeling, (5) forecasting, and (6) econometrics. Fourth, many of these planning and modeling systems are quite flexible. Changes in databases, models, and reports are easy to implement. Fifth, even if the model builders are senior programmers, econometrics, forecasting, and risk analysis are much easier to implement with one of these systems than with a scientific language.

Of course, the advantages of these planning and modeling software systems must be weighed against their costs. First, these systems are not available free of charge to the user. That is, the user must pay a fee for the use of one of these planning and modeling systems. A limited number of these systems can be licensed for use on in-house computers. These include BUDPLAN, FORESIGHT, FP-70, PSG II, and SIMPLAN. Nearly all of these systems are available on a surcharge basis on various timesharing service bureaus. Second, since the computer is doing the work of many programmers, the computer running costs will definitely be higher than say similar models programmed in FORTRAN, but the human costs should be considerably less.

SUMMARY AND CONCLUSIONS

With nearly 2,000 companies now experimenting with some form of planning model, it is not surprising to observe that many of these companies began using a particular modeling system without giving much thought to the long-run implications of the system which was selected. It is not uncommon to find one division of a company using FAL, another using PROPHIT II, and a third using a FORTRAN model running on yet a different service bureau's computer. At the same time, the corporation maintains a corporate database as well as databases for each division on the in-house computer. Corporate planning may also subscribe to one or more outside econometric forecasting services.

In other words, it is not unusual to find large companies subscribing to as many as six different modeling services with exact duplicates of the corporate database running on the in-house computer as well as on outside service bureaus.

With a little thought and careful planning, it is possible to design an integrated planning and modeling system which will satisfy corporate management as well as the management of all of the business units. Financial, marketing, and production planning models can all be developed within one system which is linked to a national econometric database. And, finally, the system can be implemented on the company's in-house computer thus eliminating outside timesharing charges and the costly duplication of databases.

In summary, time spent on the design of a company's planning and modeling system may be time well-spent.

Analysis of "natural" language discourse*

by SALLY YEATES SEDELOW
The National Science Foundation
Washington, D.C.
and
The University of Kansas at Lawrence

ABSTRACT

Referential linkage in extended language strings (multiple-sentence, paragraph, etc.) is of great interest to computer scientists, linguists, and literary scholars concerned with the analysis of discourse. In all three disciplines, semantic relationships are central to approaches to inter-sentential, inter-paragraph, and inter-supraparagraph linkings. This paper compares and contrasts some of the directions taken in current research and explores the possible utility of a general-purpose thesaurus for the construction of semantic frames of reference. The utility of such a thesaurus for measuring semantic distance between terms and thus establishing possible linkages is suggested by an experiment concerning word prefixation.

Discourse analysis is now being undertaken by research scientists in a number of disciplines. Its meaning varies from field to field and even from scientist to scientist; traditionally, it has implied the structural analysis of a relatively large number of consecutive natural language sentences, paragraphs, chapters, or larger units. (It is not restricted to the spoken word, although that is one meaning of discourse.)

As a former professor of English literature with a major interest in the analysis of language-strings of the length of *Hamlet*, or *Paradise Lost*, I have watched the growing interest in discourse of my newer colleagues in computer science and in linguistics with both genuine excitement and, it must be confessed, a frequent feeling of *déjá vu*. Thus, extended debate on the subject of "frames" at a recent workshop on theoretical issues in natural language processing sounded very like discussions over the years by literary scholars and students on topics such as "frame of reference" and "point of view." I feel that my new colleagues are, in a sense, just beginning to learn to talk but, on the other hand, perhaps in time their talk will be couched in more precisely used terms than those employed in analogous literary conversations. I certainly hope so, since such precision was for me a prime motivation in having become associated with computer scientists and linguists.

Although discourse analysis has been a primary concern of literary scholars for many years, for both linguists and computer scientists it does represent an exciting new concern. The reasons for computer science coming to this study rather late are quite obvious, but that linguists should be so slow to arrive on the scene may seem to some rather surprising. For the sake of clarification, it might be noted that for a number of decades literature and linguistics (at least as linguistics is practiced in this country) have been very much separated from each other. Linguistics departments have sometimes provided, as a service, courses on the English language, but the heart of the discipline has been elsewhere. The "elsewhere" has been a world of micro-events, such as the basic units of sound, or the basic units of grammatical structure, which were long considered to be solely syntactic. Given this concern with microunits, the outer bounds considered necessary for adequate study of such units were provided by the sentence. As I have already noted, literary scholars find it necessary and desirable to concern themselves with texts of lengths greater than a sentence; thus, the work of linguists has not been of overwhelming interest to literary scholars, and vice versa. Further, there has been relatively little of the interdisciplinary dialogue which might have pushed linguists somewhat sooner toward looking at the types of language phenomena of interest to literary scholars.

More recently, linguists have been forced, perhaps partially through their own need for new approaches to the study of language but also because of efforts to use computers for language-dependent tasks—such as machine translation—to try to place their microunits within systems which, in turn, have proved to be parts of ever larger systems. Early efforts to use the computer to provide translations produced generally unsatisfactory results partly because semantic systems had been essentially ignored by linguists. Attempts to apply the computer to other areas entailing heavy

* The views expressed in this paper are those of the author, and do not necessarily reflect any policies of the National Science Foundation.

manipulation of natural language—areas such as information retrieval and computer-assisted instruction—have clearly demonstrated both that semantics must be taken into account and that the length of a sentence does not provide an upper bound for semantic interpretation of a language string. Linguists, of course, have not been ignorant of the existence of semantics and long language strings. In effect, they simply decided to ignore these areas because they looked intractable. Currently, practical necessity and intellectual curiosity are moving linguists into discourse analysis; so that now we have a coincidence of interest among some linguists, some computer scientists, and some literary scholars (not to mention certain psychologists and other social scientists). In order to provide context for the relevance of my own research to this emerging area, I'd like to refer briefly to some forefront thinking about discourse analysis in computer science and linguistics and then show where my own work, which originated in the study of literature, fits.

At the moment, it is probably accurate to divide approaches to discourse analysis into those efforts which are concentrating upon rather short, although still multiple, sequences of sentences and those approaches which are concentrating upon texts which are much longer, even of narrative length.

Those scholars and scientists concerned with "small-scale" discourse analysis want to describe linkages between and among contiguous or near-contiguous sentences, as well as to describe linkages between a single sentence and its context; the context may be provided either by contiguous sentences or by networks of associations built up from earlier experience with reality, however perceived. A simple example of linkage between sentences is pronominal reference: both linguists and computer scientists have been concerned with this aspect of anaphora (literally, the carrying up or back to something earlier) and of cataphora (carrying forward).

Examples of computer-based natural language systems which deal with aspects of reference linkage can now be readily located. They range from examples in Winograd's block world program in which the computer responds to the command "Grasp the pyramid" by saying "I don't understand which pyramid you mean" because the particular pyramid implied by the article "the" was not specified in an earlier sentence,[1] to concerns such as those described by Barbara Deutsch at the Stanford Research Institute in her recent paper on "Establishing Context in Task-Oriented Dialogues."[2] When the SRI computer is told to "Assemble the air-compressor" and to "Begin by attaching the pump to the platform" it starts by inspecting something called focus spaces to determine whether there is more than one air compressor to worry about; it proceeds analogously when "attaching *the* pump to *the* platform."

The SRI grammar, described by Jane Robinson,[3] can also handle elliptical references between sentences, as in the following sequence: "What is the length of the surface displacement of the Lafayette? . . . What is its draft? and "What is the length of the Lafayette? . . . The Ethan Allen?" In this sequence of four sentences we have an example of anaphoric reference through the use of "its" in the second sentence and of elliptical anaphoric reference in the fourth sentence which, in its entirety, would read "What is the length of the Ethan Allen?".

Let me offer one more example of current speculation by a computer scientist about an aspect of small scale discourse analysis; in this case, the concern is with linkages among single sentences and an individual's "store" of earlier verbal and sensory experience. This example is the beginning of a fable, as told by Wallace Chafe, a linguist at Berkeley, and retold by Marvin Minsky:[4]

> *There was once a Wolf who saw a Lamb drinking at a river and wanted an excuse to eat it. For that purpose, even though he himself was upstream, he accused the Lamb of stirring up the water and keeping him from drinking . . .*

Minsky says that to understand this fable one must realize that the wolf is lying. To understand "even though" one must realize that the contamination in question doesn't move upstream. This realization in turn requires us to understand (among other things) the word "upstream," itself. Minsky then devotes several pages to indicating how his framework might help one understand some of these terms as well as the more extended meaning of these two sentences. By frame, Minsky means a

> data-structure for representing a stereotyped situation, like being in a certain kind of living room, or going to a child's birthday party. . . . For visual scene analysis the different frames of a system describe the scene from different viewpoints, and the transformations between one frame and another represent the effects of moving from place to place. For nonvisual kinds of frames, the differences between the frames of a system can represent actions, cause-effect relations or changes in metaphorical viewpoint.[4]

A little later, I want to return to this example to indicate how some of my research can help to get at "frames of reference" which, in turn, will help to delineate discourse structures where the structure depends upon the perception of semantic relationships.

For macro discourse analysis, the most relevant current work in computer science is content analytic and I will recur to it briefly later. When we turn to linguistics, we find a number of linguists—notably Robert Longacre[5,6] and Joseph Grimes[7] and students associated with them—who have become involved in the analysis of macro discourse ranging up to short narratives in length. In *The Thread of Discourse*,[7] Joseph Grimes provides a broad overview of these discourse analytic techniques, a number of which would be useful for discourses of any length. Many of the approaches to discourse analysis described by Grimes depend on too much preliminary background for use in this paper, but a brief description of two of the approaches would seem worthwhile in order to provide some sense of linguists' orientations toward this new area.

One very simple technique is span analysis. An identification span consists of a series of identifications of the same participant in a narrative—a series in which no identification is stronger than the one before it. Strength of identification is defined as a ranking that goes from proper names, to explicit descriptives like "the last speaker at the symposium," to common nouns like "the speaker," to nouns used generically like "the fellow," to pronouns like "him," to reference without identification. When a shift occurs from a weak form of identification to a stronger form, the current span is terminated and a new one begins. The spans can be shown graphically simply by taking a piece of paper, listing vertically from 1 to n the number of clauses, or phrases, and drawing vertical lines to show the length of the spans. Such spans can be used for many other characteristics of discourse such as setting, time, and place, and can reveal patterns which graphically illustrate the differences in style between one work and another, between one author and another, and so on.

I would like to mention one other discourse analytic technique described by Grimes and, in fact developed by his students, Mary Ruth Wise and Ivan Lowe.[8] This approach depends upon the roles taken by participants in a narrative. Here, roles are described by such terms as agent, experiencer, instrument, and other analogous categories drawn from case grammars, which are now of great research interest to both linguists and computer scientists. This approach to discourse analysis studies the shifting of roles among the participants in a discourse. The perception of shift is based upon relative rankings among cases such as agent, experiencer, and so on. In this system, agent is the highest ranking role and a case called essive, which is rather analogous to the existential state, is the lowest ranking.

One type of role shift is a reversal, which involves an exchange of relative ranking between two participants in a narrative. In some simple narratives, reversal is the only shift which occurs. The text tends to start out with one character as agent and the next as, say, goal, then reverses so that the second character is agent and the first has a role of lower rank. A second reversal brings the characters back into the original orientation; in some of the simple folk narratives studied by Grimes and his students, this second reversal signals the beginning of a new paragraph.

Another kind of role shift is described as a switch, in which the second and third participants undergo reversal of rank. In a number of narratives studied by Grimes and his students, the regular progression of events is carried by single operations such as a single reversal or a single switch. Whenever there are composite operations, such as a switch and reversal, or reversal and switch, there is a surprise, an interruption or a point in the narrative where things go wrong.

Grimes tries, with varying degrees of success, to provide a text-analytic operational significance to such categories as cohesion, theme, setting, background, evaluation, and many others, including a range of types of linkages between sentences. Almost all, if not all, the categories for discourse analysis he explores depend upon understanding word meanings, phrasal meanings, clausal meanings, paragraph meanings, and so on. Meaning—semantics—is the key to discourse analysis and therein lies a challenge for any efforts to use the computer for discourse analysis.

My own research on computer approaches to discourse analysis has been partly built upon training as a literary scholar. Thus I have been concerned with texts of extended lengths and I have wanted to find methodologies, procedures, and/or algorithms suitable for use with a broad range of texts. By contrast, for the most part computer scientists concerned with discourse have tended to constrain the universe of discourse with which they deal to a very small segment of reality. The reasons for such constraints are obvious when you consider the complicated problems which must be dealt with in even a very modest universe; but I have chosen not to follow this path because my own preferences and training make it uncongenial and, especially, because as a scientist I strongly prefer general solutions for natural language problems to *ad hoc* solutions.

When I began working with computers* I decided to concentrate upon structures of meaning because those are of major importance to discourse which, as I have stated before, is of central importance to most literary scholars. At that time (thirteen years ago) other people were working on parsers and I planned simply to borrow a parser from someone else when the need arose. It is doubtless a splendid testimonial to man's ingenuity as a user of language that the general-purpose parser I need is still not available, although there has been excellent research on this problem and there are parsers to which I would now like access.

My approach was to develop a set of content analytic programs which I used to look for literary themes. Themes comprise semantically-related words and form

* Much of the research described on the following pages has been supported by the Information Systems Branch, Office of Naval Research.

part of the structure of a discourse. For example, part of the structuring form of *Hamlet* is provided by a network of associations involving disease, madness, and decay, just as part of the structure of *Paradise Lost* involves a semantic network of words associated with variety and variation. I have written and talked about this research elsewhere[9-12] and I will not discuss it further here. But I do want to talk about an outcome of that research—that is, further research, on the nature of thesauri.

Thesauri now come in many forms. In the context of information retrieval, a thesaurus most often consists of a list of terms, sometimes divided into categories, relating to some specialty area. The kind of thesaurus I have in mind is not one of these special-purpose word lists, but rather a general-purpose reference work showing semantic relationships among words in a whole language. Specifically, I have in mind the English-language thesaurus, *Roget's International Thesaurus*.[13] Although no such general reference work is 'perfect,' or even nearly perfect, it does have the strong advantage of validation through time and across a culture. I argue that large corpora such as *Roget's Thesaurus* or the Merriam-Webster's dictionary should be investigated for their potential utility in dealing with semantics for all kinds of computer-based tasks. They also lend themselves to segmental exfoliation when a finer semantic mesh is needed for a specific purpose. Such potential sources of semantic information have been largely ignored by those seeking to use the computer to capture meaning in natural language. Notable exceptions to this rule are John Olney at SDC and his associates who have put *Webster's Collegiate Dictionary*[14] into computer-accessible form; among those using this version of the dictionary are Robert Simmons and Robert Amsler who, in a recent paper[15] state that "from a linguistic point of view . . . the proper source of semantic information is a large corpus of ordinary usage of the language."

One of the reasons these general purpose corpora haven't been more extensively employed is that they are difficult to use. Another is that they are difficult to get into a useable form. Part of my own research effort in recent years has been directed toward putting *Roget's International Thesaurus* into computer-accessible form. The *Thesaurus is* now in computer-accessible form although the editing hasn't been completed.

Editing is a major problem because the *Thesaurus* makes many assumptions concerning the ability of the human reader to fill in elisions, supply context, and otherwise rely on the enormous store of information in the human brain. An example of a frequently occurring type of elision is "jump up or off" in which the jump for "jump off" is omitted. An example of the dependence upon the human reader to fill in or at least understand what a possible extension of items in the *Thesaurus* might be is the frequent use of etc. to indicate the continuation of lists. Examples of such lists are "Pan-American, Pan-Pacific, Pan-Hellenic, etc.", (multiply by 5, etc.)", "hourly, daily, etc." and so on. These and other analogous problems which must be dealt with in order to use the *Thesaurus* in automated systems have been discussed elsewhere.[16,17]

Our reason for putting the *Thesaurus* into computer-accessible form was to use it for content analytic applications. In order to put it to such use, it is desirable to have a clear understanding of the structure of the *Thesaurus*; approaches to the study of the structure have been explored by a graduate student, Robert Bryan, and are described in research reports for this project (available through the Computer Science Department at the University of Kansas).[18,19] A major potential utility of the *Thesaurus* not only for content analysis but for many other efforts to use the computer to understand natural language is as a guide to measures of semantic distance. That is, the number of nodes one must pass through in the *Thesaurus* to get from one word to another, or from one concept to another, can be taken as measures of semantic distance. Again, the graph theoretic approaches outlined by Bryan are intended to facilitate research on this question. It should be borne in mind that since any given term may appear at several different nodes in the graph, semantic space in the *Thesaurus* is much more complex than the organizing tree-structure for the *Thesaurus* content might lead one to suppose.

The viability of the *Thesaurus* as a potential source of information concerning semantic distance was a central issue in an experiment we undertook with reference to prefixation. Our experiment was to see whether the *Thesaurus* might be used for determining when a particular string of alphabetic characters is serving as a prefix. The experiment is described in some detail by Sam Warfel in one of our research reports.[20] As you may know, the *Thesaurus* is organized hierarchically, with the basic text consisting of 1040 semantic categories each with a number and a label, *e.g.*, "854. lack of feelings." Each of these numbered categories is divided syntactically and semantically. For the purposes of this experiment the syntactic categories were ignored, because a preliminary investigation suggested that syntactic categories added no useful information toward the determination of prefixed words. (An obvious reason for this impression is that syntactic category membership is most often indicated by suffixes.)

To clarify the following discussion, it may be worthwhile to describe briefly the general structure of the hierarchy in the *Thesaurus* so that ensuing references to "levels" will be understandable. The 1040 categories are related at a higher level in the *Thesaurus* by the "Synopsis of Categories," which is not part of the basic text but is presented as an outline following the Preface. In this synopsis the *Thesaurus* is divided into eight classes (*e.g.*, Class Six: Intellect). Each class is divided into several labeled sub-classes indicated by Roman numerals (*e.g.*, I. Intellectual Faculties and Properties) and each sub-class is divided into labeled sub-sub-classes designated by capital letters (*e.g.*, L. Conformity to Fact). Each sub-sub-class is divided into several of the 1040 categories (*e.g.*, 515. Truth) which are numbered consecutively throughout the text. Each of the 1040 categories is subdivided into numbered 'paragraphs' of words; for example, there are twenty-two such divisions under 515. Truth, so that there are subcategories 515.1–515.22. These 'paragraphs' are, in turn, subdivided into semi-colon groups; that is, within a paragraph those words most closely related are grouped and delimited by semicolons, *e.g.*,

accuracy, correctness, rightness;
exactness, exactitude;
preciseness, precision;

Thus, in the following discussion of the value of semantic distance measures for determining prefixation, the levels referenced are as follows:

Level 1 *e.g.*, Class Six: Intellect
Level 2. *e.g.*, I. Intellectual Faculties and Properties
Level 3. *e.g.*, L. Conformity to Fact
Level 4. *e.g.*, 515. Truth
Level 5. *e.g.*, 515.3
Level 6. *e.g.*, accuracy, correctness, rightness;

Our assumption was that words occurring either in the same place in the hierarchy or very near each other in the hierarchy are more likely to be related than those further away. Therefore, it seemed possible that words having the same root and differing only by prefix would tend to fall together in the *Thesaurus,* (i.e., to be only short distances apart). In general, our sampling bore out our hypothesis. That is, words such as joy and enjoy (863), rich and enrich (835), guise and disguise (230), courage and encourage (891), occur in the same fourth level categories in the *Thesaurus* hierarchy. On the other hand, words such as *vent* and *prevent* are shown not to belong together because prevent does not occur in either the same category or any of the categories related to categories associated with the unprefixed root *vent.*

To gain a further sense of the utility of the *Thesaurus* for this purpose, we compared the use of the *Thesaurus* with that of our prefix recognizing program (which is operational and which is based on ad hoc decisions on all the words in the *Random House* dictionary).[21] We used the *Thesaurus* for the comparison of thirty-eight word pairs which had been turned up during the course of a computer run of the PREFIX program (22). PREFIX produced ten word pairs which struck me as viable pairings. Had the *Thesaurus* been used to examine those same word pairs, eight of them would have been linked together. The word pairs are shown below, with those bearing the * indicating the pairings also produced by the *Thesaurus*:

integration-disintegration
* able-enable
* courage-encourage
* danger-endanger
* doubted-undoubted
moralize-demoralize
* capacity-incapacitate
* labor-elaboration
* sequence-consequence
* valuate-evaluate

Of the ten pairs on the above list, the grouping of *moralize* and *demoralize* is at least questionable. The *Thesaurus'* judgment here may be preferable to my own when I was earlier looking at the results of the PREFIX run.

The PREFIX program grouped together three word pairs which should not have been grouped. The *Thesaurus* identified none of these words as appropriately paired:

cent-recent pare-prepares tribute-distribution

The PREFIX program produced twenty-five pairings which I judged to be helpful in some contexts (some of them very rare and esoteric) but not in others. Those pairs in the list below which would have been linked by the Thesaurus are again shown with an *:

* compass-encompass
* conceivable-inconceivable
cover-discover
* fend-defend
fluence-influence
* joy-enjoy
* junction-conjunction
ligation-obligation
mode-outmode
* ply-apply
promise-compromise
* refutable-irrefutable
* rich-enrich
* rupture-disrupt
see-foresee
* separable-inseparable
stead-instead
strict-restrict
* thinkable-unthinkable
tinction-distinction
gaged-engaged
ordinate-coordinate
vestigation-investigation
jugate-subjugate

As should be apparent from the examples above, as well as, hopefully, from any sampling others might make of the *Thesaurus,* while the *Thesaurus* is not perfect for this task, nonetheless, it is rather good. Some problems could be taken care of if inconsistencies in the organization of the *Thesaurus* were eliminated. For example, although most negation and reversal relationships are expressed in adjacent categories under the same third level headings, there are exceptions to that organizational principle. Thus, although there is a category at the fourth level in the hierarchy labeled "disintegration" there is no comparable category labeled "integration," despite the fact that there are categories labeled "order" and "disorder" as well as "continuity" and "discontinuity."

The same inconsistency is apparent with *talkativeness* and *untalkativeness* where the third level categories containing the words are quite far removed from each other, unlike other positive-negative category pairs:

(Level 2)	III. Communication of Ideas	
(Level 3)	B. Modes of Communication	M. Uncommunicativeness; Secrecy
	552. Communication	611. Uncommunicativeness
	552.4	611.2
	communicativeness, talkativeness. . . ;	taciturnity, untalkativeness. . . ;

A different problem arises with the pairs *birth-rebirth* and *born-reborn* where the first word in each pair occurs in fourth level categories related to either "beginning" or "physical birth" while the second word in each pair occurs only in fourth level categories related to the religious experience of conversion. Therefore, the words in the pairs are not judged by the program to be prefixed since the metaphoric relationship between the two uses of *birth* is not shown in the *Thesaurus.*

In order to achieve comparability in these measures of semantic distance provided by the *Thesaurus,* organizational inconsistencies in the *Thesaurus* such as revealed by this experiment would need to be coped with either through cross referencing or shifts in the structure of the *Thesaurus.* Metaphorical relationships will be more difficult to capture, although it is the case that the *Thesaurus,* unlike a dictionary, is rather strong on making some metaphorical relationships explicit. This facet of the *Thesaurus* is useful, as we shall see in an experiment with the *Thesaurus* and the few sentences from the fable about the wolf and the lamb cited earlier.

For the sake of convenience, the initial sentences of the fable I cited earlier are repeated:

> *There was once a Wolf who saw a Lamb drinking at a river and wanted an excuse to eat it. For that purpose, even though he himself was upstream, he accused the Lamb of stirring up the water and keeping him from drinking. . .*

The experiment I want to undertake here is first to use the *Thesaurus* to see what kind of contextual "frame" it can provide to facilitate understanding of the fable. When the *Thesaurus* fails, I will suggest the use of a parser or dictionary, either separately or in combination with each other or with the *Thesaurus.* Let me say, prefatorially, that this experiment, as is the case with Minsky's article on frames, ignores many difficulties that would in fact arise were one using the *Thesaurus,* as well as, for that matter, the dictionary and parser to cope with the fable. The goal in this next little exercise is to see whether there is any point in pursuing further the use of the *Thesaurus* for such a task.

Our assumption—a strong one—will be that the computer has no information about any of the words in these sentences and that, in fact, the *Thesaurus* must be used to provide the context.

You will observe that the first sentence of the fable is: "There was once a Wolf who saw a Lamb drinking at a river and wanted an excuse to eat it." A lookup of "river" in *Roget's* will locate "river" under "running water" and it will be linked with "stream" as well as with "drinking water." Thus the notion of river as drinkable emerges. The word "wolf" appears in a listing of animals. "Lamb" does not appear in that list but it is linked to "sheep" which, in turn, appears as an animal. "Sheep" is also linked to "mutton" which is linked in the *Thesaurus* index to "meat" which in turn occurs with "feed" and "dine." Given metaphors' pleasing property of frequently having some relationship to (symbolic) reality, it would probably be useful to find in the *Thesaurus* the use of "wolf" to mean "devour," which in turn occurs in the index under "eat up." Thus, as it happens, the *Thesaurus* associates wolf with eating and on the basis of this association it might be possible for a computer program to assume that the creature who "wanted an excuse to eat it" is the wolf. The lamb is not particularly associated with drinking through any references in the *Thesaurus.* Rather, a syntactic parser might be expected to work out the relationship between the lamb and drinking in the phrase "lamb drinking at a river."

The chief problem in this sentence is the identification of "it." To repeat the sentence: "There was once a wolf who saw a lamb drinking at a river and wanted an excuse to eat it." "It" could refer either to the lamb or to the river. The only route to disambiguation that looks possible to me is at the syntactic level, which might point up a parallelism between the wolf seeing an object—the lamb—and eating it. In both cases, "it" is the object of an action by the wolf and possibly through this parallelism "it" might be identified with lamb. I find no information in the *Thesaurus* that would enable a program to perform this disambiguation. It is possible to deduce from the *Thesaurus* that a lamb can be eaten if you follow a somewhat circuitous path which, under the listing of animals, provides the word "flesh" in connection with "horse flesh" and under "eating" provides the word "flesh eater." You will remember that "sheep" appeared under "animals" and that the word "lamb" was linked with "sheep." Unfortunately, there is no information in the *Thesaurus* which would suggest that a river cannot be eaten.

To the contrary, the words "drinking" and "drink" occur under the general heading of "eating," so one might conclude that the "lamb," for example, could just as well be said to be eating a river as to be drinking it. The preposition "at" in the phrase "lamb drinking at the river" might be helpful here, but it is difficult for me to see how it could solve this problem.

With reference to the word "excuse" in this sentence: the wolf "wanted an excuse to eat it"—the *Thesaurus* lists "excuse" along with "guise," "mark," "pretext" and "false pretence" and it also links "false pretender" with the expression "wolf in sheep's clothing." Syntactic analysis might be necessary to establish that it is the wolf, not the sheep, which is linked to "false pretender" in that group but, given such analysis, "excuse" is linked to "wolf."

The portion of the second sentence we have is: "For that purpose, even though he, himself, was upstream, he accused the lamb of stirring up the water and keeping him from drinking. . . ." Syntactic analysis should establish that "he, himself" is the subject of the second sentence and rules governing anaphoric reference would identify the subject, "he, himself," with the subject, "wolf," of the first sentence.

The meaning of "upstream" and its connotations for this little narrative are quite difficult to get at. As noted earlier, the *Thesaurus* is structured so that, frequently, categories having opposite meanings are juxtaposed. For example, Ascent, which contains the word "upstream" is next to Descent, which contains the word, "downstream." It would be possible to get from one category to the other by searching on the word "stream." If one looked up "upstream" in *Webster's Seventh Collegiate Dictionary,* the definition "at or toward the source of a stream" turns out not to be terribly helpful. On the other hand the definition of "downstream" is "in the direction of the flow of the stream" and flow is associated with movement both in the dictionary and in *Roget's Thesaurus.* The index of the *Thesaurus* provides many clues that "move" and "motion" entail a change of position; for example, one finds the phrases "move back" and "move forward." The dictionary, as you remember, associates movement or flow with directionality and "downstream" is shown by the thesaurus structure to be the opposite of "upstream." Further, the "even though" in this sentence implies some condition—opposite or contrary to—not consonant with one's "frame" involving "upstream."

At this point, the computer-based information processing programs might be in a position to try to produce some version of the representation suggested by Minsky in his discussion of this small segment of discourse. It might be desirable to represent the relative positions of the wolf and the lamb vis-à-vis the direction of flow of the river; and it may, indeed, be necessary to build into the computer program some sort of "primitive sense perception" to show that disturbing the water at the lamb's location doesn't affect the water at the wolf's location. This kind of "sense-perceived" knowledge, which forms the basis for much exploratory work in computer science, does not depend, initially, upon word associations but rather upon visual experience. (It may even depend also on a stored analytical model of physical processes.) Much knowledge, of course, has an *ultimate* dependence upon visual experience but this particular perception is difficult to track down to its final meaning through either thesaural associations or dictionary definitions. Given some such primitive representation of the factual situation it would then be desirable to try to contrast the verbal statement after "accuse" with the realistic representation produced by the "even though."

One can see how it might be possible to combine a parser, a thesaurus, and a dictionary, as well as, perhaps, sensory primitives to conclude that the wolf intends to pick a fight with a lamb, but I am not certain that it will be clear whether the wolf wants to eat the lamb or the river—although the suggested syntactic clues might provide some probability that the wolf would like to eat the lamb. If the syntactic information doesn't provide a satisfactory resolution, then one envisions having to provide some primitive sense perception based on the size of a river relative to that of a wolf or on what animals, perhaps even wolves, have been seen to eat, or some combination of those perceptions. The role of *Roget's* and the dictionary, obviously, would be to try to reduce markedly the number of such perceptions one would have to build into a system and to capture, instead, many such perceptions through patterns of word association which presumably reflect one's sense perceptions (and [implicit] reality models).

It is certainly clear to me, as I'm sure you will be able to guess, that an effort to get down to cases and actually program a computer to make its way through a thesaurus and dictionary, draw the appropriate inferences, and combine those inferences with syntactic and sensory information is going to be exceedingly complicated. But it needs repeated, strong emphasis that we require general, not ad hoc, solutions to these problems. I argue that one should be exploring the possible utility of these general-purpose reference works with an eye toward revising and adapting them to the needs of specific discourse analyses for the very pragmatic reason that no one seems to have the patience to construct from scratch a thesaurus or dictionary that deals with a very large segment of "reality" and, at the same time, is specifically designed for a set of "language understanding" computer programs. Further, there is at present no consensus as to what theories or procedures the computer programs should embody; therefore, there is no consensus as to how words and their semantic relationships should be represented. It is also the case that general-purpose reference works have some claim to cultural validation; such is not the case for ad hoc thesauri or dictionaries for special-purpose programs.

In summary, one can surmise, I think, how one might build upon some of the types of analyses I have suggested to use the computer to analyze discourse. For small scale, or micro discourse, syntactic parsers will be valuable, as will dictionaries and thesauri for their guides to meaning. Representation of at least visual perception "primitives" probably cannot be circumvented. I've said nothing about sound patterns, but research directed toward enabling the computer to move from the written word to spoken "output" as well as to enable the computer to understand speech is currently in progress and such research clearly has relevance for the study of patterns of sound in discourse. For macro discourse, all the procedures relevant for small-scale discourse are germane. In addition, the kinds of thematic analyses made possible by my programs are relevant for extended discourse. It is then possible to make visible patterns of thematic occurrence within a text by graphically portraying the themes at the locations in which they occur. John B. Smith of Penn State, once a student of mine, used some of my programs and some of his own to portray occurrence patterns of images in Joyce's *Portrait of the Artist*—and discovered that those moments Joyce had described as epiphanal were graphically obvious because of the coincidence of major imagery patterns at those points in the *Portrait*. Spans of setting, of identification, and of other categories to which I have alluded will eventually be amenable to computer analysis if we are able to deal with the many thorny issues related to perception of meaning that I have illustrated through examples in this paper.

The importance of computer-based discourse analysis for the many natural language applications in computer-based systems cannot be overemphasized. The long-awaited breakthroughs in information retrieval and in other application areas, such as computer-assisted instruction, which are dependent upon information retrieval (broadly defined) must await an increased algorithmic and computational capability for the analysis and generation of extended discourse. It behooves us to explore every possible avenue to full or partial solution of the many complex problems which must be solved in order to achieve computer-based discourse analysis. The implications are immense—both for the development of the non-numeric aspects of computer science and for the next major stage in the application of computers to the solution of human problems and to aiding with society's work-a-day tasks.

REFERENCES

1. Winograd, Terry, *Understanding Natural Language*, New York: Academic Press, 1972.
2. Deutsch, Barbara, *Establishing Context in Task-Oriented Dialogues*, Stanford Research Institute, Artificial Intelligence Center, Technical Note 114, September, 1975.
3. Robinson, Jane J., *A Tuneable Performance Grammar*, Stanford Research Institute, Artificial Intelligence Center, Technical Note 112, September, 1975.
4. Minsky, Marvin, "A Framework for Representing Knowledge," in Patrick Henry Winston, ed., *The Psychology of Computer Vision*, McGraw-Hill, 1975, pp. 211-177.
5. Longacre, Robert E., *Hierarchy and Universality of Discourse Constituents in New Guinea Languages: Discussion*, Georgetown University School of Languages and Linguistics, 1972.
6. Longacre, Robert E., *Philippine Languages: Discourse Paragraph and Sentence Structure*, Summer Institute of Linguistics, 1968.
7. Grimes, Joseph E., *The Thread of Discourse*, Technical Report No. 1, NSF Grant GS-3180, Cornell University, 1972.
8. Wise, Mary Ruth and Ivan Lowe, *Permutation Groups in Discourse*, Languages and Linguistics Working Papers No. 4, 12-34. Georgetown University School of Languages and Linguistics, 1972.
9. Sedelow, Sally Yeates and Walter A. Sedelow, Jr., "A Preface to Computational Stylistics," in *Computer and Literary Style*, Kent State University Press, 1966, pp. 1-13.
10. Sedelow, Sally Yeates and Walter A. Sedelow, Jr., "Categories and Procedures for Content Analysis in the Humanities," in George Gerbner, *et al.*, ed., *The Analysis of Communication Content*, John Wiley & Sons, Inc., 1969, pp. 487-499.
11. Sedelow, Sally Yeates and Walter A. Sedelow, Jr., "Stylistic Analysis," in Harold Borko, ed., *Automated Language Processing: The State of the Art*, John Wiley & Sons, Inc., 1967, pp. 181-213.
12. Sedelow, Sally Yeates, *The Narrative Method of Paradise Lost*, University Microfilms, Inc., Ann Arbor, 1961.
13. *Roget's International Thesaurus*, Third Edition, Thomas Y. Crowell Co., New York, 1962.
14. *Webster's Seventh New Collegiate Dictionary*, G. & C. Merriam Co., Springfield, Mass., 1967.
15. Simmons, Robert E. and Robert A. Amsler, "Modelling Dictionary Data," in Ralph Grishman, ed., *Directions in Artificial Intelligence, Natural Language Processing*, Courant Computer Science Report Number 7, Courant Institute of Mathematical Sciences, Computer Science Dept., New York University, August, 1975.
16. Harris, Herbert R., "The Automated Version of *Roget's International Thesaurus:* A Description with Suggestions for Future Editing," in Sally Yeates Sedelow, *et al.*, *Automated Language Analysis*, 1973-1974, pp. 6-32.
17. Sedelow, Sally Yeates, "Etc. in *Roget's International Thesaurus*," in Sally Yeates Sedelow, *et al.*, *Automated Language Analysis*, 1972-1973, pp. 35-43.
18. Bryan, Robert, "Abstract Thesauri and Graph Theory Applications to Thesaurus Research," in Sally Yeates Sedelow, *et al.*, *Automated Language Analysis*, 1972-1973, pp. 45-89.
19. Bryan, Robert, "Modeling in Thesaurus Research," in Sally Yeates Sedelow, *et al.*, *Automated Language Analysis*, 1973-1974, pp. 44-59.
20. Warfel, Sam, "The Value of a Thesaurus for Prefix Identification," in Sally Yeates Sedelow, *et al.*, *Automated Language Analysis*, 1971-1972, pp. 31-49.
21. *The Random House Dictionary of the English Language*, ed. Jess Stein, New York, 1966.
22. Sedelow, Sally Yeates, "PREFIX" in Sally Y. Sedelow, *et al.*, *Automated Language Analysis*, 1968-1969, pp. 12-25.

Computer animated film systems—A rat's nest of trade-offs

by BRUCE CORNWELL*
Consultant
Brooklyn, New York

and

KATHARINE CORNWELL*
Peat, Marwick, Mitchell & Co.
New York, New York

ABSTRACT

Selected examples of computer animated film production systems are analyzed in regard to usefulness for student projects, commercial film production and graphic research. No system seems ideal for all of these purposes and the most sophisticated systems inhibit a "collaborative" relationship with the computer, especially regarding images that relate to mathematics, science, or aesthetic spin-offs from these fields. A minimal system of batch mode computing with microfilm output is described.

The two most beautiful aspects of the computer as used in batch mode processing is that most any system is infinitely expandable from the standpoint of hardware, and it has unlimited flexibility in view of the vast range of languages that can be utilized. It is almost astounding that one can gradually expand a computer from scratch, without having to make far-reaching decisions, other than the jump from a hand-held programmable calculator to a system with peripherals, or the jump to a larger word size if one starts out with a mini computer. In contrast to this possible oversimplification, the building of a general purpose system for the production of computer animated films is practically an insoluble problem. Not only is there no ideal *final* system, but the route toward the building of a high-capacity or high-versatility system presents an obstacle course of decisions. These decisions are simplified if the system is dedicated to a single purpose, such as films for use as engineering simulation studies produced by an engineering concern with an in-house facility.

Aside from such specialized applications, there are two questions that might be explored: (1) What is a minimum, workable computer animated film system, and (2) What are the trade-offs as one expands upward? These questions are especially relevant when considering the possibility of a facility for production of computer animated films within an educational environment.

There are several insights that can be gained from an examination of the problems encountered in the selection of equipment, materials and facilities for conventional motion picture production. To begin, there is no such thing as a *general purpose* film production facility, or motion picture camera. Equally sophisticated film producers may require totally different equipment, as when shooting on a wildlife refuge, or on a sound stage. The term, *trade-off,* in the motion picture field is not a matter of differences of degree—like 10 to 25 percent in cost, portability, speed—it can mean equipment being totally incompatible for diverse tasks. Trade-off with equipment for computer animated films can be almost as drastic.

If the proposed facility is predicated toward ease of film production, eventually the filmmaking process can become child's play, and the end result may be trivial. For instance, Fortran can be used to construct an endless variety of mathematical functions as graphs. And these can be subjected to motion that is also mathematical. But, if Fortran is bypassed by the filmmaker for the option of a graphic display with light pen, the whole universe of mathematical graphics is essentially discarded. When a computer is used to execute elaborate mathematical functions—or even simple ones—this user often feels the presence of the computer as an unseen collaborator. On the other hand, freehand graphics is reputable. It was invented millenniums before mathematics. However, with the availability of interactive graphics, it is practically impossible to direct students *not* to use a tool that instantly generates dynamic results in preference to doing something the hard way. A parallel to this is a problem of students who are starting to shoot live films. Although the zoom lens is simpler and more versatile than a turret with three fixed focal length lenses, it is most difficult to

* Free lance computer animated film producers.

convince the students not to zoom in and out *ad nauseam.*

There is no absolute minimum configuration for a computer animated film system, and the minimum end of the spectrum is in almost the same state of flux as the maximum configuration end of the spectrum. However, for purposes of this discussion we would like to present for consideration a system that we have used for over ten years of experimenting, teaching, and contract production of computer films in a variety of areas. Probably it should not be called a system since there is no dedicated hardware, Figure 1. The only dedicated item is four boxes of punched cards that comprise a standard graphics software package. We are presently working with a modified version of the Integrated Graphics Software developed by the Rand Corporation. Our version is for the Stromberg Datagraphix 4060 plotter. All of the computing is done batch mode on any of several IBM-360-370 installations. The plotting has been done off-line on a number of microfilm plotters, including Datagraphix, Gould, Seaco, and most frequently the Information International Inc. FR-80. Since most microfilm plotters have the ability to emulate their competitors, there has been little need for changes in software (those subroutines that convert our vector descriptions into commands that drive the microfilm plotter).

In order to assess this minimum system within the total range of computer animated films, its throughput potential should be compared with that of several other configurations. As in discussion of live film production, it is most simple to classify a mode of film production by the end product rather than by the equipment needed to achieve that result.

The minimum system described above is reasonably effective for the production of *dynamic line images.* This would include films that demonstrate scientific as well as mathematical principles since some areas of science, particularly physics, lend themselves to mathematically controlled graphics.[1] For some obscure reason, it is practically impossible to present a dynamic mathematical structure that does not display aesthetic attributes to even the non-mathematician, especially if a music track is synchronized with the images. Probably the most significant example of this effect is in the work of John Whitney.[2] This type of film is described by curve (1) of Figure 2. It requires a great amount of planning and usually rather elaborate programs to drive the plotting software. However, the end result can be accomplished with a minimum of hardware as shown by the descent of the curve. (It should be noted that Whitney has used an interactive graphic terminal to simplify input to his driving program. This saves time.) Finally, it will be noted that this type of film requires the most time for throughput. This is not entirely discouraging in the long view. If the film is to display a unique time-motion structure, the conception, design, the writing of a driving program along with a few new subroutines that are invariably required, and the debugging of these programs—all these added together make the throughput ability of the hardware less than crucial. When this system is used for classes in computer animated film which we teach at the New School, some students choose to spend far more time at the blackboard than the computer center.[3]

A second type of film is emerging in the area of computer animated cartograms and statistical graphs.[4] This is graphed as (2) in Figure 2. Typically, this type of film displays very complex information and requires massive input data to accomplish its purpose. Once a program is operational, there is little need for the programmer to interact with the system, since the primary purpose of the film is to reveal what the data has to show. This curve (2) would also apply equally well to films that simulate traffic flow through a system of paths with symbols that represent people or vehicles.[5]

An emerging category of films could be called *simulated cinematography,* (3). The major initial thrust in this direction was made at the University of Utah

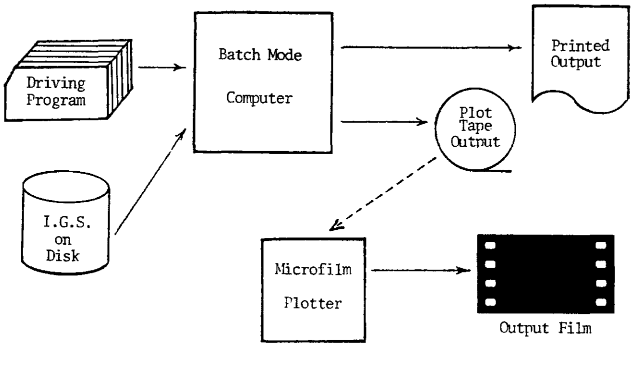


Figure 1—Flow diagram of minimum computer animated film system

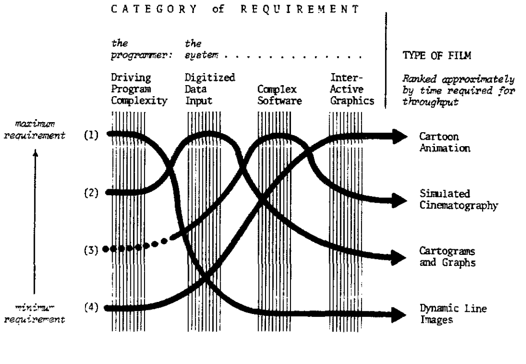


Figure 2—Comparison of computing needs required by selected types of animation

where the scheme was developed for representing curved surfaces with a continuously shaded image that would take into account the location of the light source and the reflective characteristics of the surface.[6] Or, in other words, photographs without a camera pointed at an object. The computing required to display a full frame of shaded image is vastly greater than required for line structures of considerable complexity. This is a major inhibiting factor if the object of producing motion pictures is to study the motion rather than the object in motion. Also the output film must be generated on a display that has beam intensity controllable by very small increments, unlike any of the microfilm plotters. However, considerable effort and ingenuity is being directed toward making these techniques of simulated cinematography a viable avenue of film production, especially in view of accomplishments at the University of Ohio.[7]

A fourth type of film, curve (4), is the computer animated cartoon. Probably the foremost example of this type would be the output of Computer Image Corporation.[8] This also involves a much more complex system than dynamic line images and the characteristic curve is almost a reciprocal of the *dynamic line images* curve (1). Although the computer slavishly follows the animator's guidance, it would be more fitting to consider the computer as an entire animation studio production staff, rather than a slave. This is a very sophisticated use of the computer as both a time-saving and labor-saving device. One of the main fascinations of experimenting with dynamic line images is that the aesthetic effect of the film created is often surprising. Conversely, the virtue of a cartoon animation system is its ability to deliver exactly what the designer requests.

One major omission from the graph is computer generated films that display a walk-through of three-dimensional space (usually architectural) in full color. Its characteristic curve would be a straight line through the maximum requirement in each category. Since there are only a few such systems in existence, it is hardly an optional route for building a computer animation facility from available resources. But even this area is in a state of flux with simplifications of hardware.[9]

It can be observed that the more efficient a system is at performing a given graphic task, the less feasible or encouraging it is to use that system for a diversity of generalized tasks. For instance, it would be almost ridiculous to use a system capable of cartoon animation or simulated cinematography to drive simple geometric symbols of vehicles in a traffic engineering simulation. Computer animation systems might be separated by purpose into two categories. Is the system intended to simulate other clearly established processes of motion picture production (including animation)? Or, is the system intended to offer the most direct means of producing motion graphs that describe, or are driven by, numerical processes? Or, restated, the choice is graphics that emulate graphics, or graphics that synthesize numerics.

If this comparison of various systems justifies consideration of the *minimum system* as a starting point, there are several items concerning its application that could be clarified.

Fortran seems the most commonly used language since the majority of graphic software packages are compatible with Fortran. Simscript is another very useful language, especially for engineering simulations, and it can be used to drive the graphic software, although not as simply as Fortran.

Graphic debugging can be accomplished on a line printer by a fairly simple set of subroutines, about 100 statements, which emulate the plotting software and produce strings of asterisks. Batch mode turn around can slow the throughput until the user discovers that working on several portions of a project simultaneously can leave little idle time. Another point on debugging that can effect considerable cost savings involves a modification in the graphic software. All of the subroutines that are accessed by our driving program are subject to being turned off or on by one call from the main program. This enables us to run through a lengthy sequence but plot on film only a few selected frames. This does consume extra computing time, but that cost is small in comparison with costs of debugging by means of a long plotting run.

The final component to consider is the microfilm plotter. Even if one has funds available to purchase or lease a plotter, it could be viewed as a serious misuse of funds if there were not enough footage of film output to keep the machine going at least two hours a day (figure about 100 16mm feet/hour) Furthermore, an in-house plotter demands a motion picture film processing machine, such as a Kodak Prostar, but if the film output is to be used as a master for making extra prints, or color prints, the film from the plotter should be reversal processed and that is not simple on a Prostar. And finally, unless one has a resident engineer provided by the manufacturer of the plotter, the system may be down at times less than convenient.

The alternative is to use a commercial microfilm plotting facility. Although we are located in New York City, it has proven most convenient and economical to work with a company in Los Angeles. They are located within a mile of the airport, and operate 24 hours a day, seven days a week. Our cost for this service is about 10¢ per 35mm frame. However, we often plot four separate 16mm images within the 35mm frame which reduces the cost to .025 per 16mm frame.

In conclusion, computer animated film production has one pitfall that also afflicts many would-be conventional film producers: It is quite easy to fall in love with beautiful, sophisticated and versatile equipment without first determining the objective of the undertaking toward furthering film as a communication medium. If there is any doubt about this objective, it is far safer to start small.

REFERENCES

1. Ehrlich, R., B. Cornwell and K. Cornwell, *Relativity: A Series of Computer Animated Films*, Houghton Mifflin Company, 1974.
2. Whitney, J., *Permutations*, Film Library Service, New York Museum of Modern Art, 1968.
3. Seigel, M., B. Cornwell, et al., *Batch Mode Square Dance*, (16mm color film), New School for Social Research, 1974.
4. Cornwell, B. and D. Kasik, *Unemployment 1954-1974, by Race*, The Graphic Social Reporting Project, Bureau of Social Science Research, 1975.
5. Joline, E., *The Application of Computer Drawn Motion Pictures to Urban Transportation Concepts*, Personal Rapid Transit III, 1976.
6. Joline, E., "Application of Computer Drawn Films for Validation and Visualization of Airport Simulations," Winter Simulation Conference, December 1971.
7. Watkins, G., *Real Time Visible Surface Algorithm*, University of Utah Technical Reports UTC-CSC-70-101, July 1970.
8. Csuri, C., "Computer Animation," *SIGGRAPH Proceedings*, 1975, SIGGRAPH Conference on Computer Graphics and Interactive Techniques.
9. Csuri, C., "Real Time Computer Animation," IFIPS Congress, Stockholm, 1974.
10. Holman, D. and L. Holman, "A Better Mousetrap, the CAESAR," Filmmakers Newsletter, February 1975.
11. Goldstein, R. and R. Nagel, "3-D Visual Simulation," *Simulation*, January 1971, Simulation Councils, Inc.
12. Eastman, J. and J. Staudhammer, "Computer Display of Colored 3-D Object Images," *Proceedings of the 2nd Annual Symposium on Computer Architecture*, p. 23-7, 1975.

Speakeasy—A window into a computer*

by STAN COHEN
Argonne National Laboratory
Argonne, Illinois

ABSTRACT

Speakeasy is a system that enables a user to harness the power of a computer as a tool for problem solving. This paper describes the structure of that system and demonstrates some of its capabilities. The examples show some of the power that results when relatively straightforward graphical facilities are added to this general purpose system.

INTRODUCTION

A modern computer complex is an extremely fast information processor with a vast amount of stored information available to it. Machines and their associated software are becoming ever faster, ever larger and ever more complex. This growth in capability brings with it increasing responsibility to provide the normal user with adequate access to available information. Unfortunately a sense of isolation from advances in computer science is common in many disciplines and results in the people most in need of the capabilities provided by a computer being forced to rely on secondary sources for solutions to their problems or to ignore advances entirely and to program using first generation computer techniques.

One answer to this problem of information exchange can be found within the structure of the computer itself. A generalized modular system that is designed to access and operate with libraries of stored information can make advanced algorithms and general information available to users in a sensible way. Properly structured, such a system acts as an interface between user and the computer, matching the user's needs to capabilities available to him. Inadequacies in such a system are answered by adding new capabilities to satisfy existing needs. A system designed for growth provides the environment to answer the needs of users; it can at the same time provide researchers in computational science with an audience for their work.

This paper describes a system of this type that has been in use and under development for over a decade. Originally designed as a tool for research scientists, the generalized and easily used structure of this processor has now led to its acceptance by a large and varied user community. Although Speakeasy[1-4] has been in existence for many years, it is the widespread availability of time-sharing systems such a TSO and VM/CMS that accounts for the recent rapid growth in acceptance. The Speakeasy processor is currently available for use on IBM 360 or 370 computers operating under OS, TSO, VM/CMS or MTS. It has been translated for use on FACOM-230 and PDP-10 computers. Translations to several other computer systems are now being carried out.

There is an active user group involving some 70 different installations and over a thousand users. The user communities already include physical scientists, engineers, econometricians, government agencies and universities. The growth in the capabilities of Speakeasy in recent years is partially in response to the diverse nature of its user group and partially due to contributions from it.

This paper is intended as a general description of the Speakeasy system. It is divided into two major sections. The first describes the structure of the system. The general form of this system should be of interest to a computer scientist since it is here that the extensibility and growth capabilities reside. The second section is a sampling of the capabilities found in actual operational versions of the system. The features illustrated are only a few of those that make this system valuable to its users. They show that a truly user oriented system can at the same time be a powerful one.

THE OVERALL DESCRIPTION

The structure of the Speakeasy system is illustrated in Figure 1. The system consists of a language processor that interprets the requests of the user and breaks them down into basic syntactical components. The language structure is a straightforward one modeled on conventional mathematics. A scratch-pad storage facility available to the processor enables it to maintain transient information for the duration of the

* Work performed under the auspices of the U.S. Energy Research and Development Administration.

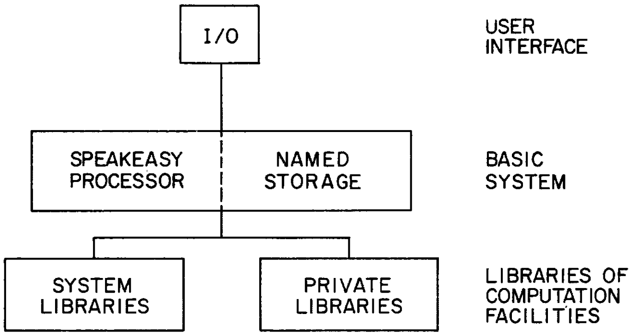


Figure 1—A schematic view of the Speakeasy system. The bulk of the computational facilities are contained in the modules that make up the libraries attached to the system

particular application. The processor and this scratch pad provide the mechanism for a user to define and address structured objects that represent transient information in the tasks being assigned to the system.

Libraries of pre-compiled routines are attached to the processor to provide the system with its working vocabulary. When a specific operator is requested of the system, all of the libraries attached to the system are searched. If the requested operator is found, it is dynamically linked to the system so that it may perform its function. In most cases such operations involve operands previously defined and saved in the transient scratch-pad area. New results are also saved there and are therefore available for later use by other operations.

This operator-operand formulation with a wide variety of operators on structured objects is the heart of the Speakeasy system. The ability to define structured objects such as arrays, vectors and matrices makes it meaningful to develop a large set of operators tailored to specific data structures. The use of attached libraries makes it possible to provide an ever increasing set of such operators without creating a system of immense size and complexity.

The modularity of this system, the use of libraries and the simplicity of its control language combine to provide a powerful system that is easily used and capable of sustained growth.

Design of the language

The Speakeasy system can be viewed as a repository. The communication language for the system (also called Speakeasy) is intended to make this repository usable. In designing the language and its syntactical rules the foremost considerations were naturalness, ease of use and tolerance for trivial mistakes. The basic specification for the language is a simple one. If a request by a user is unambiguous and looks correct to him then it should be accepted by the system.

This design philosophy is somewhat different from that of normal computer languages that provides features to enable the user to exploit the capabilities of the machinery itself. Such languages are specified so that a large number of decisions that relate only indirectly to the calculation must be made by the programmer. To give just a few examples, fixed point versus floating numbers, dimension statements for quantities that occur only as intermediate results, and input and output formats are concepts only indirectly related to the statement of the problem itself. Frequently such relatively trivial specifications constitute the bulk of a program; the parts related to the calculation are but a small part of the material written by the user. It is the volume of this extraneous information that accounts for many of the errors encountered in conventional programming.

In Speakeasy most of such decisions are considered to be part of the internal functions of the system. The user formulates his problem in a brief and natural manner, the system translates this formulation into an executable form, relieving the user of as many trivial decisions as possible. In fact, the Speakeasy language specifications bear little relationship to the structure of a computer and only very indirectly reflect the form of calculation as executed in the computer.

The Speakeasy language is designed to operate with structured objects. Scalars, arrays of numbers, mathematical vectors and true matrices are among the many classes of structured objects that a user may define and use in his calculation. Implicit algebraic rules that are class dependent provide a variety of tools for formulating calculations while operating on defined objects as single entities. Complete matrix algebra is provided in a natural notation. This combined with the array processing capabilities, enables one to write a directive program that does not involve the loops and branches that make up most of the computational steps in conventional programs.

The availability of structured objects means that an operator-operand language with great richness can be developed (one of the limitations of usual languages is that the scalar structure of the languages limit functional values to scalars). Each of the operators in Speakeasy is designed as a self-contained module whose operations are dependent upon the structure of its operands. In this way all of the decisions normally necessary in invoking routines in conventional programming techniques are internally contained in the operators of the system. Each operator is clearly definable in terms of what it does and contains many checks to see that it is being properly applied. By placing decision processes such as these within the system the user is relieved of most of the mundane parts of programming. The user can therefore concentrate his effort on formulating the overall logic of his problem and still be assured that the large number of trivial decisions are being properly made.

The overall language designed around an operator-operand concept combined with conventional algebraic tools is powerful, easily learned and easily used. It is logically complete and is extensible in both the types of structure supported and the operators available for manipulating objects.

The transient scratch-pad—named storage

A special dynamic storage scheme is an important component of the Speakeasy system. This storage facility provides the mechanism for defining and retrieving the structured objects discussed previously. In this scheme each defined object has associated with it a complete description of its structure along with particular values for its elements. Most importantly, the object and its descriptive information can be referenced by name. The name is all that is needed to locate any defined object, to determine everything about its structure and to use it in a calculation.

Named Storage[5] was developed for use with calculations in nuclear physics.[1] It is this storage technique that led to the development of Speakeasy. Individual modules of the Speakeasy system (the operators available to the language processor) are designed to operate on objects defined in Named Storage and to produce new objects there. The extreme modularity of the Speakeasy system is a consequence of this rather straightforward storage scheme.

Each defined object in Named Storage has descriptive information appended to it which designates its class, the type of data in its elements and its dimensionality. The allowed ranges for these designators has been made large so that new data types, new classes, etc. can be supported in later developments. Since no computational capabilities are contained in the storage package it is possible to extend the capabilities of Speakeasy entirely through additions to the operators attached to the system.

The libraries—linkules

In most computer systems each new addition brings with it increases in complexity, more overhead and a larger processor. Even with a very modular system this remains the case. The benefit of each addition must be weighed against the consequence that it will have on the overall system performance. It is therefore unlikely that a feature of great benefit to only a few users will ever be adopted if it degrades the system's use for others.

The library orientation of the Speakeasy system solves this problem in a particularly interesting way. The use of attached libraries containing operators such as those described earlier provides a system with easily expandable capability in a way that neither constrains the growth nor increases the basic structure of the system. Operators that are dynamically attached to the processor, called linkules,[3] provide the means for adding any desired capability for those who need it, without others even being aware of its existence. Private libraries of linkules can be used to give each user community a processor tailored to its own needs.

The freedom provided by a library oriented system is obvious. Growth by the accumulation of new information is automatic. Maintenance is particularly easy since replacement, modification and additions are made on members of libraries and not on the processor itself.

Each new operator added to the system brings with it capabilities that are enhanced by the presence of other operators in the system. A system such as Speakeasy thus reaches a critical stage where the interrelationship between the operators begins to make itself felt. After this threshold is reached the growth of the capabilities of the system are often based on finding new ways to interconnect existing facilities. Speakeasy has passed that threshold and it is no longer possible to clearly define the limits of the capabilities of the existing system.

The user interface

The user's first introduction to a system such as Speakeasy can be through a variety of devices. It is important that a system such as Speakeasy functions equally well for each type of device and that it be able to exploit the particular capabilities of each. In this system this is accomplished by isolating all input and output to specific components of the system, designing interfaces to classes of devices, and providing other general facilities that can be selectively made available to users of specific devices on demand.

This ability to interface correctly to every type of terminal in a time sharing environment is of particular psychological importance since the intent is to make this system appear natural to the casual user. If the introductory session with the processor is spent describing how a particular device is used or why a multitude of peculiar keys are used then it is unlikely that the user will ever be convinced of the naturalness of the language.

The currently operational versions of Speakeasy can be used with card readers, printers, plotters, ASCII terminals, IBM 2741's, Tektronix 4000 series graphics terminals, or any combinations of these devices. In each case the adaptation is one in which the device is natural to use and one where all of its hardware capabilities can be exploited.

EXAMPLES OF SPEAKEASY

Speakeasy is best demonstrated in a time sharing environment, for it is there that the ease of use and natural form of the language becomes most obvious. Complete novices can begin to carry out calculations with only a few minutes instruction. Built in documentation and teaching aids enable such users to learn about the wide variety of capabilities within the system and to soon begin to utilize the system as a tool in their daily work. As their demands grow they find within the system a wide variety of facilities that can be merged together into an extremely powerful tool.

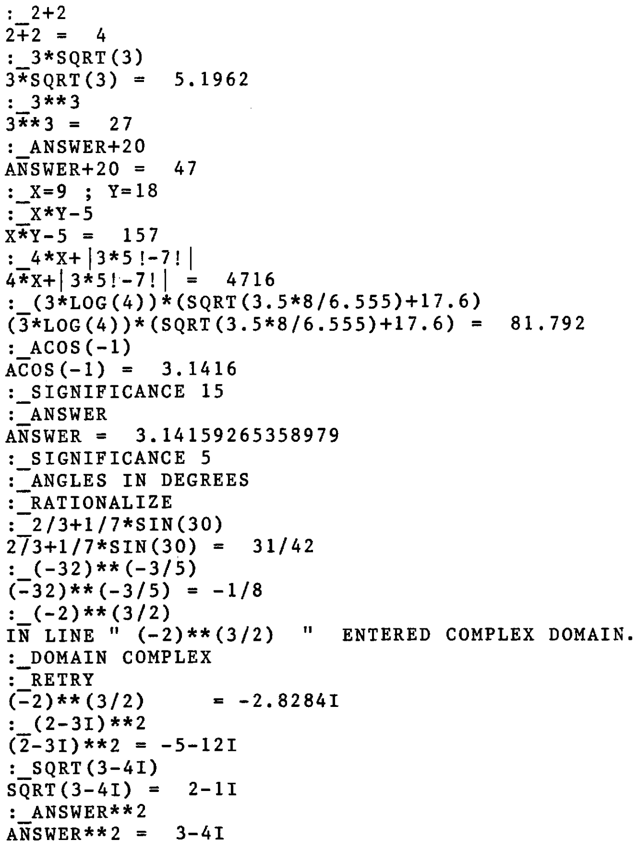
It is obviously not possible in this short paper to convey the feeling associated with an interactive system. Neither is it possible to demonstrate more than a few of the capabilities of this large system. No attempt is made, for instance, to illustrate the use of stored programs or of the editor that is available to construct such programs. Neither is there any discussion of the means of storing and retrieving information. The examples given here must therefore be viewed as a demonstration of only a few of the available features and perhaps as a tantalizing taste of what can be found within the system.

Examples of the arithmetic capabilities

The bulk of the computational capabilities in this system are related to numerics. Many of these facilities were provided by simply designing interfaces to existing Fortran coded mathematical subroutines and functions. Every attempt is made to find the best available technique and to provide as sensible and as general a facility for the users as is possible. It is in this way that the novice computer user is able to make use of advanced computational techniques.*

The system contains the basic operations of numerical calculus, statistics, matrix algebra and most of the usual special functions. Many of the operations are available for both real and complex numbers.

Figure 2 shows the processor being used as a desk calculator. The two characters :_ are the prompt symbols for the manual mode of Speakeasy. The user's request is typed on this remainder of the line. A carriage return indicates that the request is complete and that processing should take place. If the request elicits a response it is almost immediate and it is followed by a new prompt. A special implied print convention echoes simple input requests and prints the result. This result is also given the name ANSWER so that it may be used in later calculations. Users may define variables by straightforward assignment statements such as those illustrated. Standard mathematics notation is clearly demonstrated by the use of absolute value signs and factorials, and by the normal hierarchy for the evaluation of involved expressions.



```
:_2+2
2+2 =  4
:_3*SQRT(3)
3*SQRT(3) =  5.1962
:_3**3
3**3 =  27
:_ANSWER+20
ANSWER+20 =  47
:_X=9 ; Y=18
:_X*Y-5
X*Y-5 =  157
:_4*X+|3*5!-7!|
4*X+|3*5!-7!| =  4716
:_(3*LOG(4))*(SQRT(3.5*8/6.555)+17.6)
(3*LOG(4))*(SQRT(3.5*8/6.555)+17.6) =  81.792
:_ACOS(-1)
ACOS(-1) =  3.1416
:_SIGNIFICANCE 15
:_ANSWER
ANSWER =  3.14159265358979
:_SIGNIFICANCE 5
:_ANGLES IN DEGREES
:_RATIONALIZE
:_2/3+1/7*SIN(30)
2/3+1/7*SIN(30) =  31/42
:_(-32)**(-3/5)
(-32)**(-3/5) = -1/8
:_(-2)**(3/2)
IN LINE " (-2)**(3/2)  "  ENTERED COMPLEX DOMAIN.
:_DOMAIN COMPLEX
:_RETRY
(-2)**(3/2)      = -2.8284I
:_(2-3I)**2
(2-3I)**2 = -5-12I
:_SQRT(3-4I)
SQRT(3-4I) =  2-1I
:_ANSWER**2
ANSWER**2 =  3-4I
```

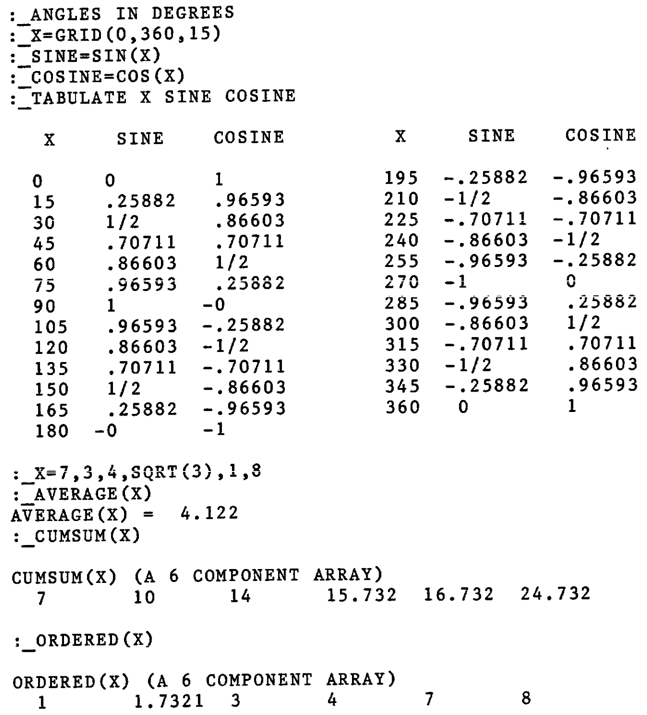
Figure 2—A sample of the use of Speakeasy in the manual or desk calculator mode of operation. Note the natural form of the directive language

Figure 3 illustrates the definition and use of arrays of numbers. It is of course not possible to show more than a few of the available capabilities. The compactness and the direct form of the language should be apparent. The rather natural and readable form of output should also be noted. It should be realized that this output form is the default form. Further tailoring is of course possible.

Figure 4 shows the means for defining and operating with matrices. The computational power that is hidden within these few statements should be apparent to those familiar with this field of numerical analysis. Some of the available tailoring commands are also illustrated here to demonstrate the flexibility of the system.

In Speakeasy two dimensional arrays and matrices belong to different families. The algebraic rules for operating on such objects are different. In Figure 5 some of the operations shown in the previous figure are repeated using arrays instead of matrices. The differences are apparent.

* For example, EISPACK is a general purpose eigen-analysis package. In Speakeasy the words EIGENVALUE and EIGENVECTOR are used to invoke this package. Decisions relating to the particular choice of path through EISPACK are made by the linkule interface. The novice user is thus provided with a powerful computational package in a simple way.



```
:_ANGLES IN DEGREES
:_X=GRID(0,360,15)
:_SINE=SIN(X)
:_COSINE=COS(X)
:_TABULATE X SINE COSINE

  X      SINE     COSINE        X      SINE     COSINE

  0     0        1             195   -.25882   -.96593
 15     .25882   .96593        210   -1/2      -.86603
 30     1/2      .86603        225   -.70711   -.70711
 45     .70711   .70711        240   -.86603   -1/2
 60     .86603   1/2           255   -.96593   -.25882
 75     .96593   .25882        270   -1         0
 90     1       -0             285   -.96593    .25882
105     .96593  -.25882        300   -.86603   1/2
120     .86603  -1/2           315   -.70711   .70711
135     .70711  -.70711        330   -1/2      .86603
150     1/2     -.86603        345   -.25882   .96593
165     .25882  -.96593        360    0        1
180    -0       -1

:_X=7,3,4,SQRT(3),1,8
:_AVERAGE(X)
AVERAGE(X) =  4.122
:_CUMSUM(X)

CUMSUM(X) (A 6 COMPONENT ARRAY)
 7       10       14       15.732   16.732   24.732

:_ORDERED(X)

ORDERED(X) (A 6 COMPONENT ARRAY)
 1       1.7321  3        4        7        8
```

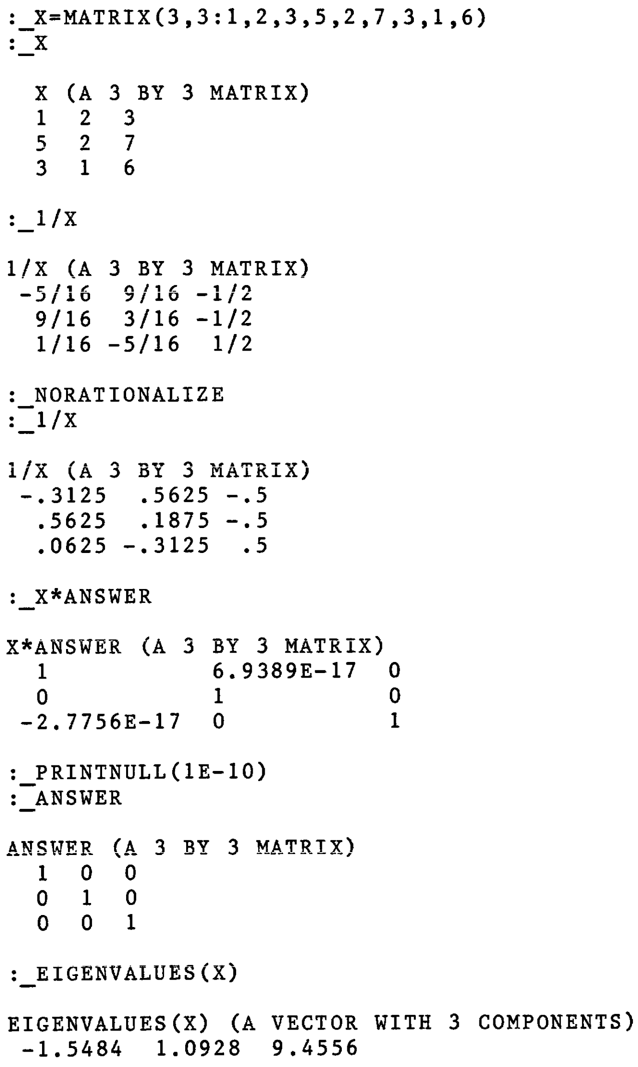
Figure 3—The compact form of the Speakeasy language is shown here. There are a large number of functions such as CUMSUM and AVERAGE. The explicit looping so common in other languages is rarely used in Speakeasy

Documentation

Any computer system must be properly documented in order that it be usable. Documentation for a major system is not an easy task. If approached solely by conventional means it would be even more difficult in a system like Speakeasy that is designed for growth. Any printed documentation would be outdated before it was published. Fortunately, the system itself is capable of providing facilities that not only supply the documentation but do so in a way that is guaranteed to be self-sustained.

A library of documents referred to as the Speakeasy HELP documents is attached to the system. This library is addressed by a linkule in the normal library. Each word, concept or facility available in the system is described in a member of the HELP document library. The library is designed as a tree structure so that a user probing the library in an interactive session is led quickly to the specific concept of interest.

Figures 6-8 illustrate the HELP documents and show how the tree structure provides quick access to information about a specific operation. Each of the over 500 documents in this library is available in a similar way. The intent of these short documents is to inform the user of the operational definition of each word. No attempt is made to explain specifics of the techniques used. A second library containing larger documents is available for that purpose and is equally easily accessed.



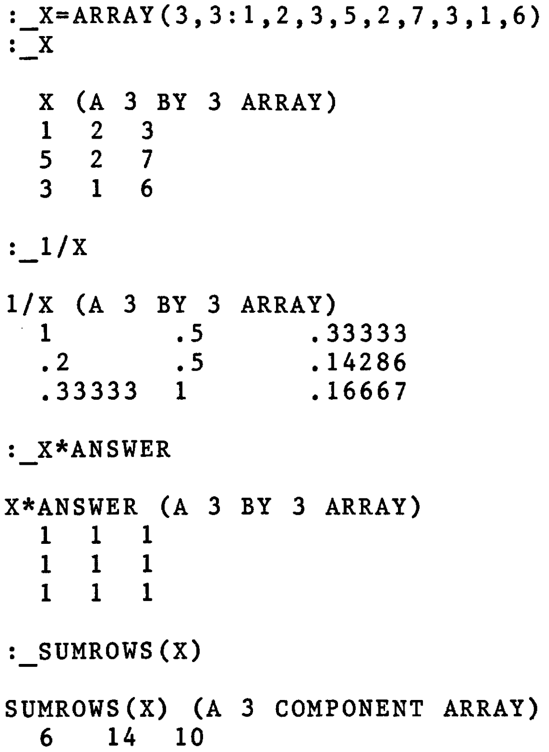
```
:_X=MATRIX(3,3:1,2,3,5,2,7,3,1,6)
:_X

  X (A 3 BY 3 MATRIX)
  1  2  3
  5  2  7
  3  1  6

:_1/X

1/X (A 3 BY 3 MATRIX)
 -5/16  9/16 -1/2
  9/16  3/16 -1/2
  1/16 -5/16  1/2

:_NORATIONALIZE
:_1/X

1/X (A 3 BY 3 MATRIX)
 -.3125  .5625 -.5
  .5625  .1875 -.5
  .0625 -.3125  .5

:_X*ANSWER

X*ANSWER (A 3 BY 3 MATRIX)
 1            6.9389E-17  0
 0            1           0
 -2.7756E-17  0           1

:_PRINTNULL(1E-10)
:_ANSWER

ANSWER (A 3 BY 3 MATRIX)
  1  0  0
  0  1  0
  0  0  1

:_EIGENVALUES(X)

EIGENVALUES(X) (A VECTOR WITH 3 COMPONENTS)
 -1.5484  1.0928  9.4556
```

Figure 4—Built-in matrix algebra is available in a natural way

A complete teaching facility is also built into the library of the standard system. A series of tutorial sessions provide a novice with step by step instructions on the use of special facilities. One of these sessions is an introduction to Speakeasy graphics. A few pages of this are shown in Figure 9.

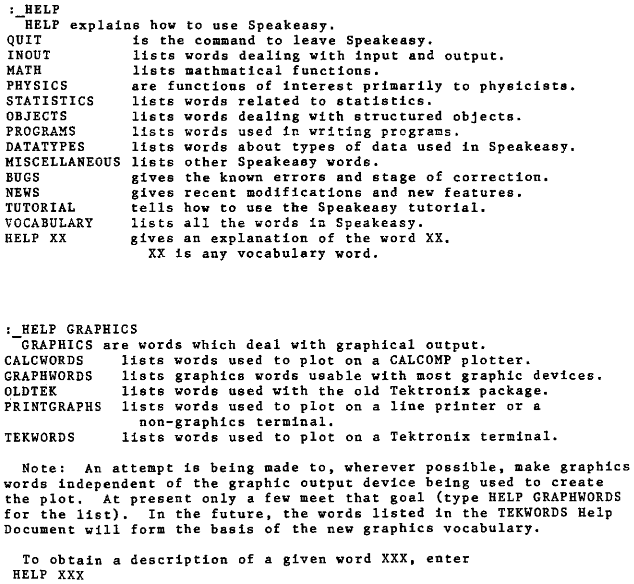
Interactive graphics

A graphics display terminal in a time-sharing environment greatly increases the potentials for true interactive computing. This is particularly evident for the exploratory types of computation that are so common in many problem solving and research environments. Because large amounts of information can be rapidly and sensibly displayed it is possible for the user to quickly interpret the effects of various choices.



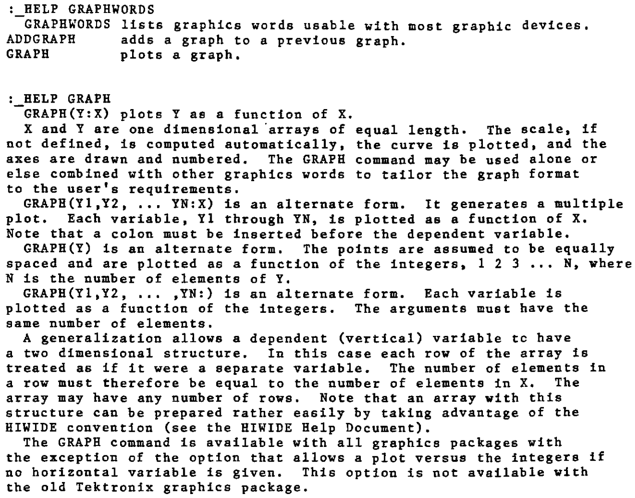
```
:_X=ARRAY(3,3:1,2,3,5,2,7,3,1,6)
:_X

  X (A 3 BY 3 ARRAY)
  1  2  3
  5  2  7
  3  1  6

:_1/X

1/X (A 3 BY 3 ARRAY)
  1        .5      .33333
  .2       .5      .14286
  .33333  1        .16667

:_X*ANSWER

X*ANSWER (A 3 BY 3 ARRAY)
  1  1  1
  1  1  1
  1  1  1

:_SUMROWS(X)

SUMROWS(X) (A 3 COMPONENT ARRAY)
  6   14  10
```

Figure 5—The algebraic operations in Speakeasy are class dependent. The array algebra in these examples should be contrasted to the algebra shown in Figure 4



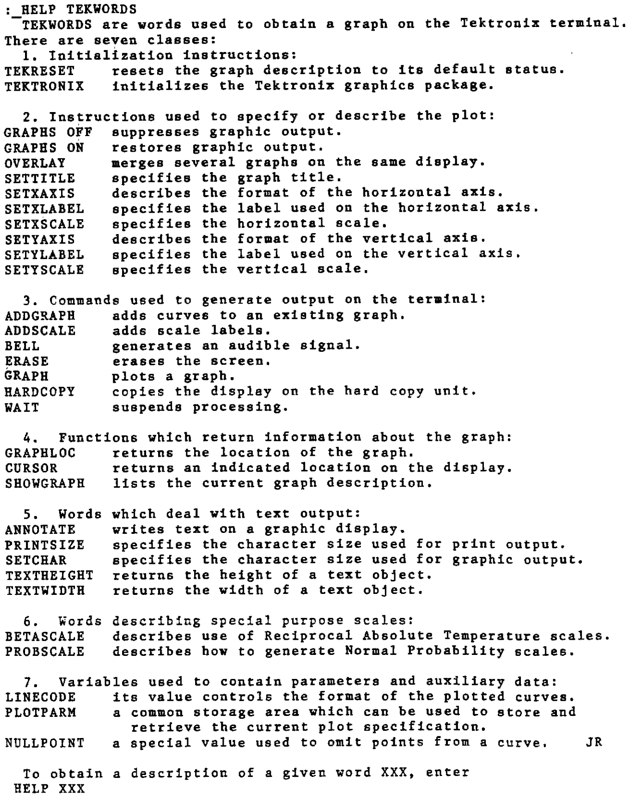
```
:_HELP
 HELP explains how to use Speakeasy.
QUIT          is the command to leave Speakeasy.
INOUT         lists words dealing with input and output.
MATH          lists mathmatical functions.
PHYSICS       are functions of interest primarily to physicists.
STATISTICS    lists words related to statistics.
OBJECTS       lists words dealing with structured objects.
PROGRAMS      lists words used in writing programs.
DATATYPES     lists words about types of data used in Speakeasy.
MISCELLANEOUS lists other Speakeasy words.
BUGS          gives the known errors and stage of correction.
NEWS          gives recent modifications and new features.
TUTORIAL      tells how to use the Speakeasy tutorial.
VOCABULARY    lists all the words in Speakeasy.
HELP XX       gives an explanation of the word XX.
                XX is any vocabulary word.


:_HELP GRAPHICS
 GRAPHICS are words which deal with graphical output.
CALCWORDS    lists words used to plot on a CALCOMP plotter.
GRAPHWORDS   lists graphics words usable with most graphic devices.
OLDTEK       lists words used with the old Tektronix package.
PRINTGRAPHS  lists words used to plot on a line printer or a
               non-graphics terminal.
TEKWORDS     lists words used to plot on a Tektronix terminal.

 Note: An attempt is being made to, wherever possible, make graphics
words independent of the graphic output device being used to create
the plot. At present only a few meet that goal (type HELP GRAPHWORDS
for the list). In the future, the words listed in the TEKWORDS Help
Document will form the basis of the new graphics vocabulary.

 To obtain a description of a given word XXX, enter
 HELP XXX
```

Figure 6—Every word known to the Speakeasy processor is described by a short HELP document. These documents are arranged in a tree structure, part of which is shown here



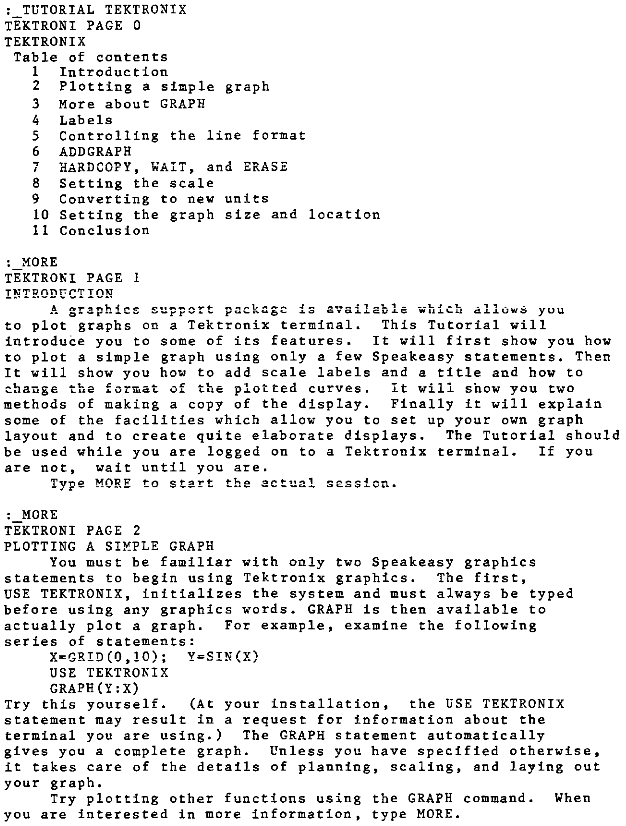
```
:_HELP GRAPHWORDS
 GRAPHWORDS lists graphics words usable with most graphic devices.
ADDGRAPH     adds a graph to a previous graph.
GRAPH        plots a graph.


:_HELP GRAPH
 GRAPH(Y:X) plots Y as a function of X.
 X and Y are one dimensional arrays of equal length.  The scale, if
not defined, is computed automatically, the curve is plotted, and the
axes are drawn and numbered.  The GRAPH command may be used alone or
else combined with other graphics words to tailor the graph format
to the user's requirements.
 GRAPH(Y1,Y2, ... YN:X) is an alternate form.  It generates a multiple
plot.  Each variable, Y1 through YN, is plotted as a function of X.
Note that a colon must be inserted before the dependent variable.
 GRAPH(Y) is an alternate form.  The points are assumed to be equally
spaced and are plotted as a function of the integers, 1 2 3 ... N, where
N is the number of elements of Y.
 GRAPH(Y1,Y2, ... ,YN:) is an alternate form.  Each variable is
plotted as a function of the integers.  The arguments must have the
same number of elements.
 A generalization allows a dependent (vertical) variable tc have
a two dimensional structure.  In this case each row of the array is
treated as if it were a separate variable.  The number of elements in
a row must therefore be equal to the number of elements in X.  The
array may have any number of rows.  Note that an array with this
structure can be prepared rather easily by taking advantage of the
HIWIDE convention (see the HIWIDE Help Document).
 The GRAPH command is available with all graphics packages with
the exception of the option that allows a plot versus the integers if
no horizontal variable is given.  This option is not available with
the old Tektronix graphics package.
```

Figure 7—A continuation of the tree search shown in Figure 6. These documents are short and supply operational definitions



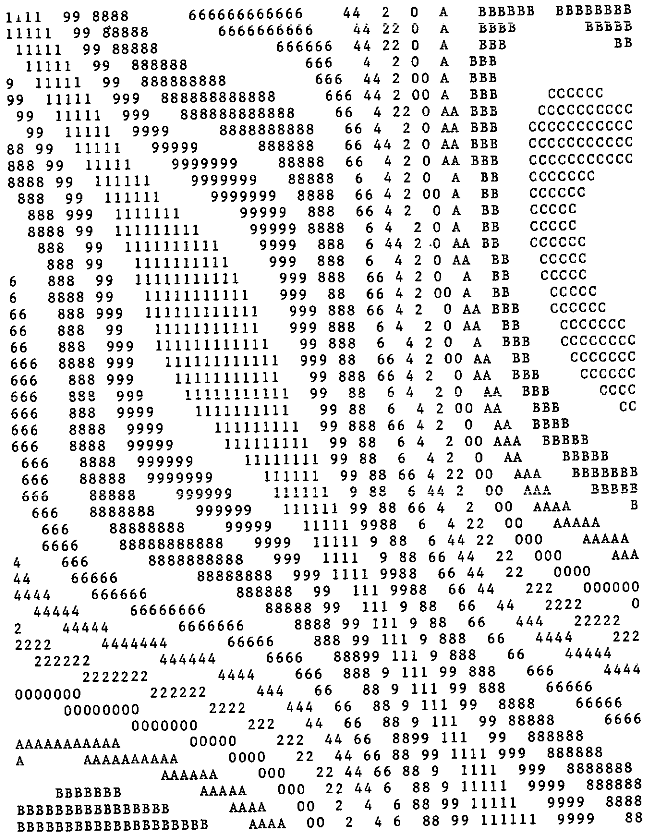
```
:_HELP TEKWORDS
 TEKWORDS are words used to obtain a graph on the Tektronix terminal.
There are seven classes:
 1. Initialization instructions:
TEKRESET    resets the graph description to its default status.
TEKTRONIX   initializes the Tektronix graphics package.

 2. Instructions used to specify or describe the plot:
GRAPHS OFF  suppresses graphic output.
GRAPHS ON   restores graphic output.
OVERLAY     merges several graphs on the same display.
SETTITLE    specifies the graph title.
SETXAXIS    describes the format of the horizontal axis.
SETXLABEL   specifies the label used on the horizontal axis.
SETXSCALE   specifies the horizontal scale.
SETYAXIS    describes the format of the vertical axis.
SETYLABEL   specifies the label used on the vertical axis.
SETYSCALE   specifies the vertical scale.

 3. Commands used to generate output on the terminal:
ADDGRAPH    adds curves to an existing graph.
ADDSCALE    adds scale labels.
BELL        generates an audible signal.
ERASE       erases the screen.
GRAPH       plots a graph.
HARDCOPY    copies the display on the hard copy unit.
WAIT        suspends processing.

 4. Functions which return information about the graph:
GRAPHLOC    returns the location of the graph.
CURSOR      returns an indicated location on the display.
SHOWGRAPH   lists the current graph description.

 5. Words which deal with text output:
ANNOTATE    writes text on a graphic display.
PRINTSIZE   specifies the character size used for print output.
SETCHAR     specifies the character size used for graphic output.
TEXTHEIGHT  returns the height of a text object.
TEXTWIDTH   returns the width of a text object.

 6. Words describing special purpose scales:
BETASCALE   describes use of Reciprocal Absolute Temperature scales.
PROBSCALE   describes how to generate Normal Probability scales.

 7. Variables used to contain parameters and auxiliary data:
LINECODE    its value controls the format of the plotted curves.
PLOTPARM    a common storage area which can be used to store and
              retrieve the current plot specification.
NULLPOINT   a special value used to omit points from a curve.    JR

 To obtain a description of a given word XXX, enter
 HELP XXX
```

Figure 8—The list of HELP documents that relate to the Tektronix graphics package



```
:_TUTORIAL TEKTRONIX
TEKTRONI PAGE 0
TEKTRONIX
 Table of contents
   1  Introduction
   2  Plotting a simple graph
   3  More about GRAPH
   4  Labels
   5  Controlling the line format
   6  ADDGRAPH
   7  HARDCOPY, WAIT, and ERASE
   8  Setting the scale
   9  Converting to new units
   10 Setting the graph size and location
   11 Conclusion

:_MORE
TEKTRONI PAGE 1
INTRODUCTION
     A graphics support package is available which allows you
to plot graphs on a Tektronix terminal.  This Tutorial will
introduce you to some of its features.  It will first show you how
to plot a simple graph using only a few Speakeasy statements. Then
It will show you how to add scale labels and a title and how to
change the format of the plotted curves.  It will show you two
methods of making a copy of the display.  Finally it will explain
some of the facilities which allow you to set up your own graph
layout and to create quite elaborate displays.  The Tutorial should
be used while you are logged on to a Tektronix terminal.  If you
are not,  wait until you are.
     Type MORE to start the actual session.

:_MORE
TEKTRONI PAGE 2
PLOTTING A SIMPLE GRAPH
     You must be familiar with only two Speakeasy graphics
statements to begin using Tektronix graphics.  The first,
USE TEKTRONIX, initializes the system and must always be typed
before using any graphics words. GRAPH is then available to
actually plot a graph.  For example, examine the following
series of statements:
     X=GRID(0,10);  Y=SIN(X)
     USE TEKTRONIX
     GRAPH(Y:X)
Try this yourself.  (At your installation,  the USE TEKTRONIX
statement may result in a request for information about the
terminal you are using.)  The GRAPH statement automatically
gives you a complete graph.  Unless you have specified otherwise,
it takes care of the details of planning, scaling, and laying out
your graph.
     Try plotting other functions using the GRAPH command.  When
you are interested in more information, type MORE.
```

Figure 9—The start of the tutorial session on the Tektronix graphics package



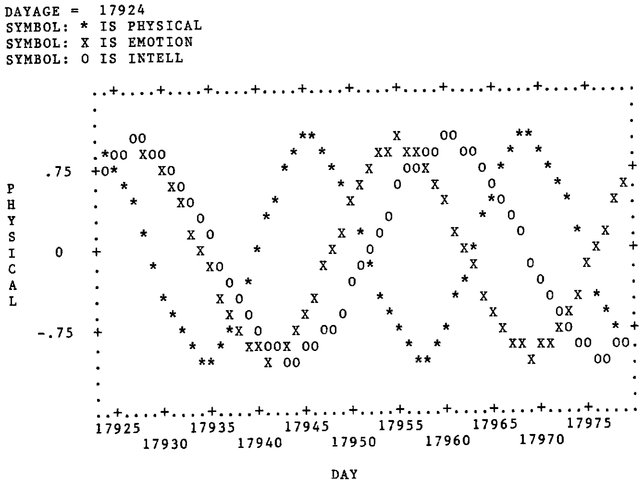
```
1111  99 8888      666666666666    44  2  0  A   BBBBBB  BBBBBBBB
11111  99 88888       6666666666    44 22 0  A   BBBB        BBBBB
 11111  99 88888            666666  44 22 0  A   BBB            BB
  11111  99  888888            666   4  2 0  A  BBB
9  11111  99  888888888         666  44 2 00 A  BBB
99  11111  999  888888888888     666 44 2 00 A  BBB     CCCCCC
 99  11111  999  888888888888     66  4 22 0 AA BBB    CCCCCCCCCC
  99  11111  9999    8888888888    66 4  2 0 AA BBB   CCCCCCCCCCC
88 99  11111  99999       888888   66 44 2 0 AA BBB   CCCCCCCCCCC
888 99  11111   9999999    88888  66   4 2 0 AA BBB   CCCCCCCCCCC
8888 99  111111   9999999   88888  6   4 2 0  A  BB   CCCCCCC
 888  99  111111    9999999  8888  66 4 2 00 A  BB    CCCCCC
  888 999  1111111     99999  888  66 4 2  0 A  BB    CCCCC
  8888 99  111111111    99999 8888  6 4  2 0 A  BB    CCCCC
   888  99  1111111111    9999  888  6 44 2 .0 AA BB   CCCCCC
    888 99  1111111111    999  888  6  4 2 0 AA  BB   CCCCC
6   888  99  11111111111    999 888  66 4 2 0  A  BB   CCCCC
6   8888 99   11111111111   999  88  66 4 2 00 A  BB    CCCCC
66   888 999  111111111111   999 888 66 4 2  0 AA BBB   CCCCCC
66   888  99   11111111111   999 888  6 4  2 0 AA  BB    CCCCCCC
66   888  999  111111111111   99 888  6  4 2 0  A  BBB   CCCCCCCC
666  8888 999   111111111111  999 88  66 4 2 00 AA  BB    CCCCCCC
666   888 999    11111111111   99 888 66 4 2  0 AA  BBB   CCCCCC
666   888  999    11111111111  99  88  6 4  2 0  AA  BBB     CCCC
666   888  9999    1111111111   99 88  6  4 2 00 AA  BBB       CC
666   8888  9999     111111111  99 888 66 4 2  0  AA  BBBB
666   8888  99999     111111111  99 88  6 4  2 00 AAA  BBBBB
 666   8888  999999     11111111 99 88  6  4 2  0  AA   BBBBB
 666   88888  9999999     111111  99 88 66 4 22 00  AAA   BBBBBBB
 666    88888    999999    111111  9 88  6 44 2  00  AAA    BBBBB
  666   8888888    999999   111111 99 88 66 4  2  00  AAAA      B
   666    88888888    99999   11111 9988  6  4 22  00   AAAAA
   6666    88888888888   9999  11111 9 88  6 44 22  000    AAAAA
4    666      8888888888   999  1111  9 88 66 44  22  000     AAA
44    66666         88888888  999 1111 9988  66 44  22   0000
4444    666666          888888  99  111 9988  66 44   222   000000
  44444    66666666        88888 99  111 9 88  66  44   2222     0
2   44444        6666666      8888 99 111 9 88  66   444   22222
2222    4444444      66666    888 99 111 9 888  66   4444    222
  222222      444444      6666   88899 111 9 888   66    44444
       2222222      4444    666  888 9 111 99 888    666     4444
0000000     222222    444   66   88 9 111 99 888     66666
     00000000      2222    444  66   88 9 111 99   8888     66666
            0000000     222   44  66   88 9 111   99 88888      6666
AAAAAAAAAAA      00000     222  44 66  8899 111   99  888888
A      AAAAAAAAAA     0000   22   44 66 88 99 1111 999  888888
              AAAAAA    000   22 44 66 88 9  1111  999  8888888
    BBBBBBB       AAAAA   000  22 44 6  88 9 11111  9999  888888
BBBBBBBBBBBBBBB       AAAA   00  2  4  6 88 99 11111   9999  8888
BBBBBBBBBBBBBBBBBBB     AAAA  00  2  4 6  88 99 111111  9999   88
```

Figure 10—A sample of the output of the contour plotter for non-graphics terminals

He is then in a position to conjecture sensible alternative problems. It is in such situations that the wealth of available facilities combined with the concise and natural directed language of Speakeasy plays its most powerful role. The system provides an environment that matches closely the needs of such a user. He may quickly formulate and carry out alternative calculations and see their results. The flexibility and power of the system becomes most apparent when the results of one calculation lead to ideas that can be tested on the spot by applying new techniques built from tools provided by the system.

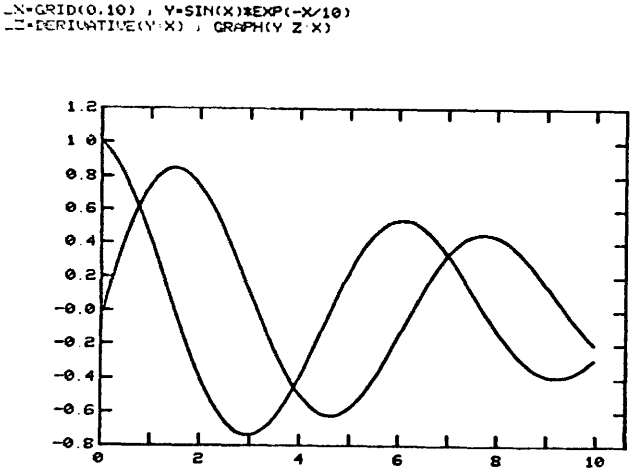
The graphical capabilities in Speakeasy are straightforward ones. They are designed to provide the user with the means of displaying computed functions in easily interpreted ways. The discussion here is limited to the facilities currently available for display terminals such as the Tektronix 4012 or 4014. Alternatives exist for other graphical devices and crude graphics are even available on printers and terminals limited to character displays. Figure 10 shows a simple contour plot that can be obtained on such a device. Figure 11 shows a sample of graphical output on a printer.

Perhaps the most dramatic example is illustrated in Figure 12. This example is the result of entering the



```
DAYAGE =  17924
SYMBOL: * IS PHYSICAL
SYMBOL: X IS EMOTION
SYMBOL: O IS INTELL

          ..+....+....+....+....+....+....+....+....+....+.....
          .                                                        .
          .                                                        .
          .   00               **         X    00      **          .
          .*00 X00             *  *    XX XX00   00   *  *         .
      .75 +0*   X0            *    *   X   00X     0 *    *        +
P         .  *   X0                 * X   0   X     0      *      X.
H         .   *   X0         *       X         X   *0       *    X .
Y         .          0      *            0         *  0            .
S         .    *    X 0             X * 0       X      0     *  X  .
I       0 +          X     *       X   0         X*            X   +
C         .     *     X0          X   0*          X     0     X    .
A         .             0 *          0           *       0         .
L         .      *     X 0       X      *       *  X      0  X *   .
          .       *     X 0     X   0    *          X      0X   *  .
     -.75 +        *    *X 0   X  00      *    *     X     X0    * +
          .         *  *  XX00X 00         *  *       XX XX  00  00.
          .          **     X 00            **          X      00  .
          .                                                        .
          .                                                        .
          ..+....+....+....+....+....+....+....+....+....+.....
          17925     17935     17945     17955     17965     17975
               17930     17940     17950     17960     17970

                                DAY
```

Figure 11—It is often desirable to produce graphical output on a non-graphics terminal. This is an example of one form of such output available in Speakeasy



```
_X=GRID(0.10) ; Y=SIN(X)*EXP(-X/10)
_Z=DERIVATIVE(Y:X) ; GRAPH(Y Z:X)
```

Figure 12—A simple graph and the statements that produced it

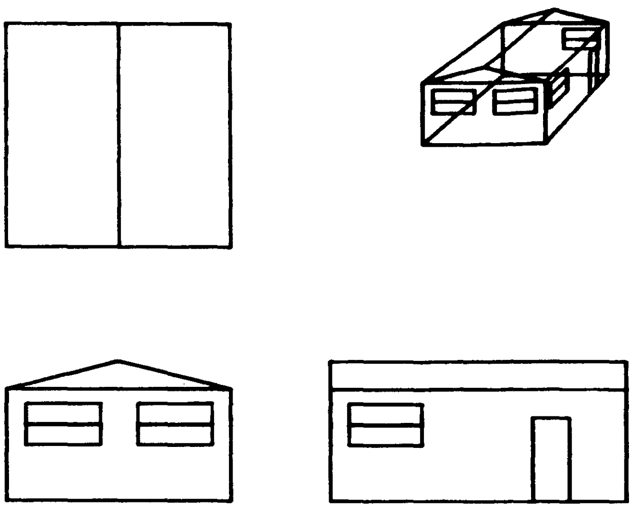


Figure 14—Another use of the three-dimensional package showing its use for a variety of projections of an object

four statements:

X=GRID(0,10)
Y=SIN (X) *EXP (−X/10)
Z=DERIVATIVE (Y:X)
GRAPH (Y,Z : X)

This example makes use of a device-independent graphics package being developed by John H. Reynolds at Comsat Laboratory that is distributed as part of the standard Speakeasy system.

As indicated by this example, automatic scaling and grid generation are provided as a default. A large number of control facilities enable the user to further tailor the graphical output to his specific needs. Figure 13 is a graph that illustrates some of the other features of this package. This graph also shows some of the statistical capabilities of the system. It was contributed by R. A. Stack of The First National Bank of Chicago.

Basic three-dimensional graphics have recently been added to the facilities available[6]. The approach taken has been to define a three-dimensional object by creating a matrix of its vertex coordinates and then describing the edges through a connectivity array involving these verticies. The power of the arithmetic capabili-

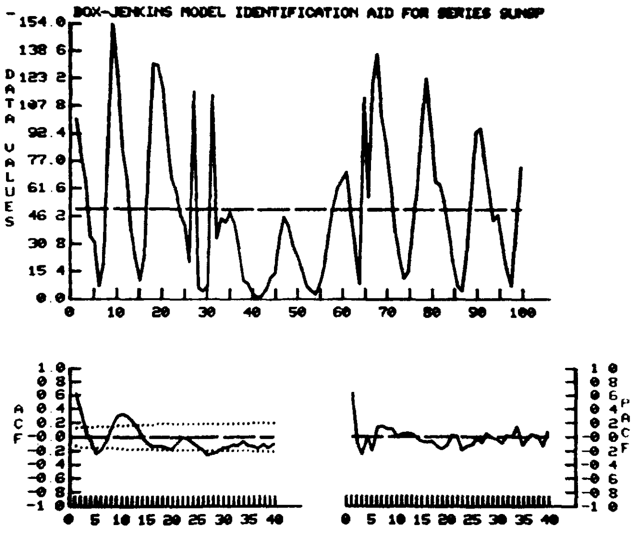


Figure 13—A set of graphs that were produced by a Speakeasy program

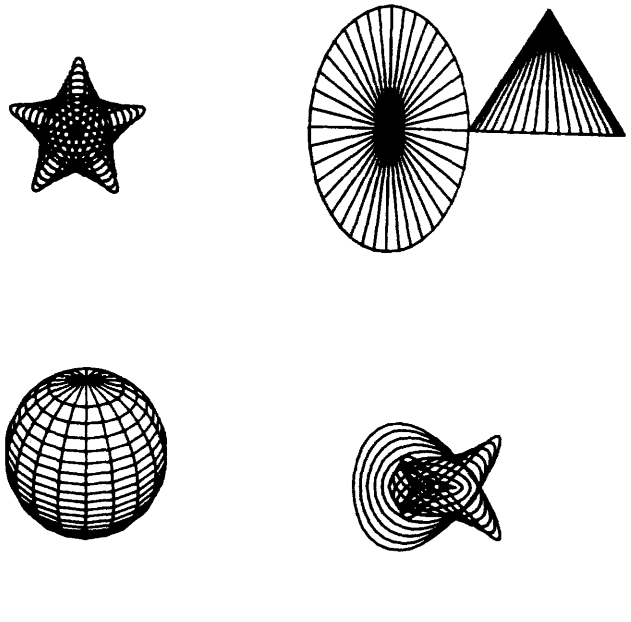


Figure 15—A sampling of some of the capabilities that are available in a system that combines general mathematical tools with simple graphics

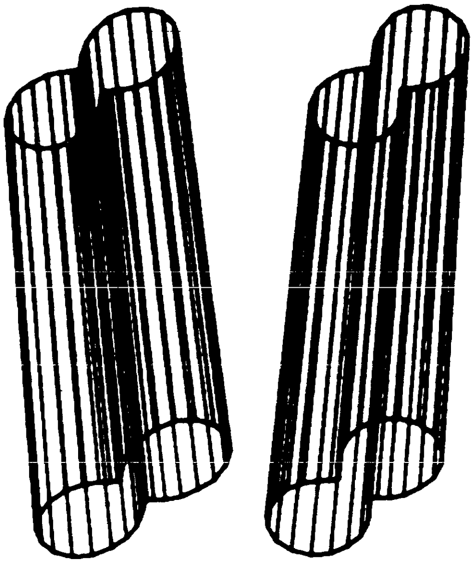


Figure 16.—An example of a steroscopic pairs produced by the three-dimensional graphics package

ties of Speakeasy can be used to manipulate the defined objects. The built-in matrix algebra, in particular, makes it easy to rotate, move or distort objects. Several special linkules were provided to help in the generation of objects made up of several components. The simple command DRAW3D is used to create the visual display. Options for perspective projections are also provided. Figures 14 through 16 show some of the possibilities.

REFERENCES

1. Cohen, S., "The DELPHI-SPEAKEASY System," *Computer Phys. Commun.* 2(1), pp. 1-10, January 1971.
2. Cohen, S., "Speakeasy: RAKUGO," *First USA-Japan Computer Conference Proceedings*, Tokyo, 1972.
3. Cohen, S., *The Speakeasy-3 Reference Manual,* Argonne National Laboratory Report ANL-8000, May 1973.
4. Cohen, S., "Speakeasy," *Sigplan Notices* 9(4), pp. 118-126, April 1974.
5. Cohen, S., *Named Storage*, Argonne National Laboratory Report ANL-7021, April 1964.
6. Blackmond, K., "Three Dimensional Graphics in Speakeasy," unpublished, July 1975.

A case study of a young child doing turtle graphics in LOGO*

by CYNTHIA J. SOLOMON
Boston University
Boston, Massachusetts

and

SEYMOUR PAPERT**
Massachusetts Institute of Technology
Cambridge, Massachusetts

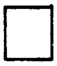
ABSTRACT

This paper explores some important issues with regard to using computers in education. It probes into the question of what programming ideas and projects will engage young children. In particular, a seven year old child's involvement in turtle graphics is presented as a case study.

This paper describes and comments on the experience of a young child in the MIT AI-LOGO Lab where she was involved in talking in LOGO to a display turtle and a PDP-11/45 computer. The child, a second grader, spent several hours on a consecutive Saturday and Sunday engaged in interesting debugging sessions. She worked long and hard. Why she could do so and why the experience was so interesting is partially explained by looking at her past experiences. In mid-January, the year before, when she was a first grader, she and I started working together learning about turtles and their world and thus explored turtle graphics. She visited twice a week for a month, staying from ½ to ¾ of an hour. We continued to meet, but less regularly, until the end of April. During that time she learned to talk to the display turtle. She learned the LOGO turtle commands like CLEARSCREEN (CS), FORWARD (FD), RIGHT (RT), LEFT (LT), PENDOWN (PD), PENUP (PU); and she learned to use them to make up her own commands for the turtle.

My goal for her first year had been for her to understand procedures both by using them and constructing them. She was given the following kind of experience. She made the turtle draw something by a series of direct commands. She would then think of a name for the picture (or piece of picture) and teach that word to the computer. To help her in this construction I wrote down the commands as she debugged them. When she "taught" the procedure to the computer she would either read the commands as I had written them or I would read them to her. Then she would try out or "run" the procedure and see if there were any bugs.

The kinds of debugging situations Est encountered varied but I was always ready to intervene in case the situation became unresolvable for her. I presented Est with the same kind of materials and projects as I developed for older children. What I expected to see with a young child was a clearer indication of where bugs in the material and ideas lay, e.g., what ideas are hard to grasp and what ideas can be understood if presented in a crisper manner or imbedded in better situations. From the sessions with Lin, another first grader I had worked with quite extensively, I developed techniques and aids which have helped older children get into turtle work, and subprocedurization, debugging, anthropomorphizing.

Let me back off a bit here and explain what preparations I had made. To aid kids in defining procedures and to exploit the idea of teaching things to the computer I provided a procedure called TEACH. This command was used instead of LOGO's TO for defining procedures. TEACH requested a name for the procedure to be defined and then asked for each instruction of the procedure by saying "STEP 10:" etc. until the child typed "END". Thus line numbers were assigned to each instruction starting at 10 in increments of 10. I also prepared procedures for making squares, circles and pieces of circle. They require inputs, which allow their size to be varied. The child also had the choice of using either RSQUARE or LSQUARE, RCIRCLE or LCIRCLE, RARC or LARC. For example,  could be drawn by RSQUARE or

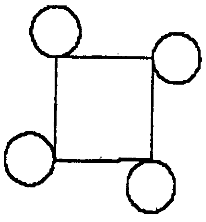
* The research described in this paper was conducted at Massachusetts Institute of Technology in the Artificial Intelligence Laboratory's LOGO GROUP. Under the support of the National Institute of Education Grant No. NIE-G-74-0012, and of the National Science Foundation Grant No. EC-40708X.

** Comments by Dr. Papert.

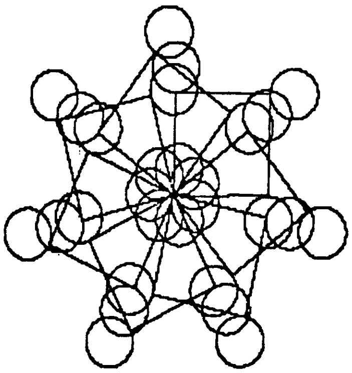
LSQUARE; only the turtle's starting and stopping states indicate which procedure should be used. All of these procedures were treated as primitive commands. Nonetheless, most children will teach the turtle to make a square or circle of fixed size by using FORWARD and RIGHT, thereby understanding the turtle's behavior in the process.

The turtle became more than a drawing device. It was a creature with certain behaviors which are interesting to study and might help us understand ourselves. The turtle lives on a display screen. Its initial state is in the middle of the screen with its nose pointing north or at 0 degrees. We can change its state by telling it to move FORWARD some number of units or turn RIGHT some number of degrees. We have marked the screen with 4 differently colored labels, NORTH, EAST, SOUTH and WEST. So we begin to build up a description of the turtle which is outside of the words provided by the LOGO language. Some of these we use to form a meta-language while others we turn into LOGO commands.

The previous remarks are meant to be background to the core of this paper, which is a picture of Est's two day interaction in the turtle-LOGO world after a break of almost six months. When Est arrived there was an initial bit of awkwardness. Her father was with her and wanted to see what kinds of things she would be doing. The work area was drastically changed. And Est wore a patch on her left eye. Her work from six months ago was in her workspace. I suggested she show her father her flower, a rather spectacular piece of opportunism. She exclaimed, "Oh yes you say CB seven times for this." She did it, her father satisfied at having seen something and reassured that she could see, left. We then abandoned last year's work and proceeded to reinvestigate the turtle's behavior. She remembered turtle commands in their abbreviated form like CS, FD, RT, LT, BK and also TEACH. She had difficulty remembering how to execute commands. That is she forgot to press the CR button and she also forgot to space between words. Last year (in anticipation of the Lebel keyboards) we had marked the CR key DOIT and thus the metaphor of "tell the computer or turtle to DO IT." Unfortunately the key was no longer so marked. But Est developed an interesting way out as a result. This little anecdote will be discussed later.



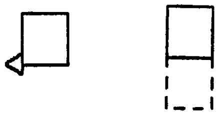
CB



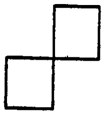
FLOWER

I had not made a firm plan because I wanted to see what she remembered, how she had changed, what the atmosphere was like. I didn't want to burden her with last year's experience. I had wanted to start off fresh and she too wanted that so I cleaned out her workspace. Intellectually we'd build on what she knew but we wouldn't examine last year's work. (Have you ever tried to understand a program you wrote six months ago!) I asked her to make the turtle draw a square or a box. She preferred to think of it as a box. (Last year she had written a procedure called BOX.) I told her in review that RT 90 headed the turtle from NORTH to EAST.

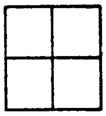
She made a square. Using TEACH (another result of work with Lin) she defined FOX, her box. I helped her by writing down what she did and reading it out to her. But now I wanted to make her independent. I posed the following problem. Make another FOX under the first like this:

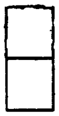


The turtle drew FOX and its stopping state was 90 degrees left of its starting state. This made the problem harder, more distracting. Est kept producing

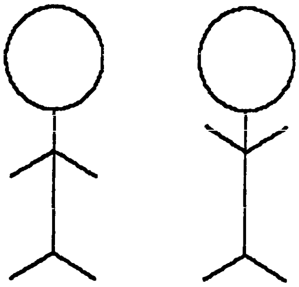


The turtle's actions upside down!

or this 

but this  was hard. I sat down and talked with her about the turtle's nose when it started and when it stopped. I said "Maybe it would be easier if the turtle ended the same way as it started." So we changed FOX. She did find it easier.

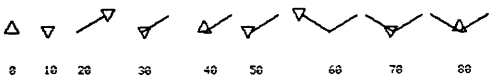
Our next project was a man, a stick figure man. I drew it and said, "What's that?" "A man" she said.



We debugged it together facing the difficulties of this being the same as this , but upside down. I picked this kind of figure rather than one with different arms and legs because it is easier and harder. There are fewer parts making it easier, but the idea of representing arms and legs by the same procedure is a bit jarring the first time. It is, of course, part of exploiting the subprocedure game. Another new idea she encountered was having to relocate the turtle's starting state to accommodate the man's head. We had to back the turtle up 90 units before running the man procedure.

We taught the computer how to draw the LEGS/ARMS. First she taught the computer to EH which caused the turtle to draw this .

```
TO EH
10 RT 60    She knew she could choose another angle
20 FD 100
30 BK 100
40 LT 60    we emphasized the 2 part process
50 LT 60
60 FD 100
70 BK 100
```



```
80 RT 60
END
```

Now Est forgot for a moment that our plan had been to turn EH upside down. When she ran EH she complained it wasn't making legs. This is interesting because it often happens with older kids as well. The idea of rotating objects to make them be different is contrasted here with rotating objects to understand they don't change. But here we look at the object differently. This shape can only be arms, not legs, but rotated like this , it can be arms and legs.

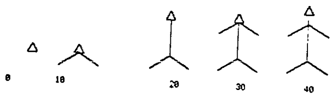
She taught D.

```
TO D
10 RT 180   last year she would have said RT 90 RT
                90
20 EH
30 RT 180
END
```

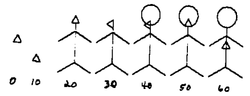
She then put the pieces together and added a neck. This year she chose numbers like 150 and 40. Last year she would have picked 43, 49, etc. or 90.

When it was time to make a head there was very little screen space left. I saw it coming and began suggesting teaching the body parts to the computer. When she ran into the difficulty we were already making plans to deal with it. She taught UH and then before running it she backed the turtle 90 units. (This time she used 90. Was it because BK was not as familiar as FD?) When we made the head there were rotation decisions and then size decisions. She had forgotten the effect of RCIRCLE/LCIRCLE's input, it was the radius not the diameter, but one buggy drawing was a sufficient reminder. Here were her procedures, UH and RUTH. By the way, using D for both arms and legs worked out well and surprised me.

```
TO UH
10 D
20 FD 150
30 D
40 FD 40
END
```



```
TO RUTH
10 BACK 90
20 UH
30 LEFT 90
40 RCIRCLE 50
50 RIGHT 90
60 BACK 100    We took 90 away from 150 and added
  40.
END
```



The way we worked followed last year's pattern:

1. Draw something on paper.

2. Draw "it" on the screen using direct commands, including subprocedures already taught. The quoted it ("it") means that we are opportunistic. If something better turns up as we draw we might change our goal.

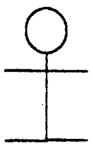
3. Now we teach the computer to do what we just did. (This is the step on which I shall concentrate in the next pages.) So the model for the learner is:

```
10  Do something
20  Teach the computer to do it
```

In more detail we could add:

5 Plan it first
15 Think about how you do it
25 Think about why it didn't quite work . . . debug

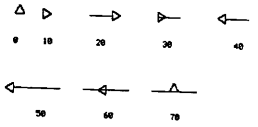
Next I asked Est to make another man without destroying RUTH. We discussed how it differed from RUTH. This one was to be MARIE. Est really wanted to make MARIE and worked hard at it. What she could and what she could not do by herself revealed some interesting patterns I have often seen and talked about but which have not been discussed explicitly. So let's look in detail at her progress.



Marie

Remember RUTH used UH to make the body. UH ran D to make the arms and the legs. To make MARIE we want to replace UH by a new procedure (Est eventually called it S), but first she needed to replace D. She called her new procedure K.

```
TO K
10 RT 90
20 FD 100
30 BK 100
40 RT 180
50 FD 100
60 BK 100
70 RT 90
END
```



Est does not do this directly. Perhaps doing so is too "formal" for her age. Perhaps she was following a pattern I had set up last year. Whatever the reason her way was to "be MARIE herself", i.e., she would give the instructions as direct commands which would later be taught using TEACH. This is how she did everything. But there is more than one way to do this. So back to details.

I left Est entirely on her own. She worked for some X minutes and eventually produced the result she wanted on the screen. This obviously shows a mastery of FD, RT etc. as well as an ability to organize her work. But now comes a difficulty often seen in children this age. She has on the paper in front of her all the commands for MARIE. She knows how to use TEACH and certainly could have typed it all in (this she had done before). But her immediate goal was different. She set herself the sub-goal of "teaching the legs", i.e., of making a subprocedure K, which she would later incorporate into MARIE. But she blocks here. She seems to find it hard to isolate just the instructions she needs for this. Why? Is this a quirk of my teaching or something deep? Seymour says it looks like a "figure ground problem", "structure dependent perception", "reversibility" and like what J. Bamberger sees in children's descriptions of clapping. I don't know, but it feels like a real problem.

I watched from a distance and eventually decided to intervene following a principle of allowing children enough success soon enough to make the fight worth while. It needed hardly any intervention. In cases like this it usually doesn't need much. Often it is sufficient to say: "Ok, let's do it together." But then all I do is read to her what she sees on her paper. Another technique that gets the same result is to write, or have the child write, on a piece of paper just what she has typed to the computer. Why do these subtle things help? Because it is a trivial problem? Maybe. But perhaps also because it is a deep problem related to what psychologists might call attention and what we might call the control process of sub-procedure management. Anyway, it needed very little intervention and she was off on the track and soon Marie worked.

Research issues: understand this phenomenon, get better at observing just what intervention works and why, track the progress of a child over longer periods.

SOME GENERAL OBSERVATIONS

1. Compared with last year Est could work much more independently. (More than half a year is a big piece of her life!)

2. Her work with RCIRCLE and LCIRCLE would have been easier if the inputs were diameter (which "exists") rather than radius (which is about a non-existent point called center). Although later she made a design which was understandable because the input was the radius.

3. She had trouble remembering to ↵ . This she cured by playing this game with the computer: after typing an instruction she would say, firmly and dramatically "DO IT" while hitting the CR key. She knew what "game she was playing". There was no trace of my manipulating her. On the contrary she manipulated herself. We'll see another good example in day 2 of how she is able to set up a deliberate strategy of programming around her own perceived bugs.

I set up the model for DO IT and feel that the way I did it (the rhythm, the degree of "reality" and also playfulness, etc.) made a very big difference as to whether this kind of thing works. On Day 2 she invented similar techniques of her own. So perhaps my suggestion took only because it was a kind of thing she does spontaneously. Big research issue!!

Day 2—Preamble:

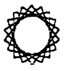
Again I had no detailed plan except the general idea of making an animation in which RUTH and MARIE alternate to give a jumping effect. I had written a new aid to help in snapping pictures called TS whose effect is to save SNAPs. But when I used it with Est I hadn't tried it out and it had a bug! I really got flustered by that. Lesson: don't use undebugged stuff, but if you do don't get flustered.

To warm up I showed Est POLY. She played with inputs. I showed her POLY 50 90. Then I asked her to change the shape. She tried POLY 50 80. She didn't want to call it a star (a 9 pointed star), then she tried POLY 50 40, POLY 50 60. Then she tried POLY 20 30, and POLY 20 20. She remarked that the last two were different sized circles. Finally she tried POLY 20 100 followed by POLY 50 100. She had to struggle with the idea that the 2 figures were the same shape. The first one was very tiny. So she tried POLY 100 100. That she thought could be the same as POLY 50 100, but still the size of the figures did bring into question whether the shape was the same.

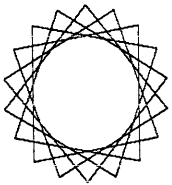
My intention on the day before was to make an animation using RUTH and MARIE. We continued to work on this scheme after I fixed a TS bug. When I discussed what we had to do:

```
DISPLAY :RUTH
WIPECLEAN
DISPLAY :MARIE
WIPECLEAN
```

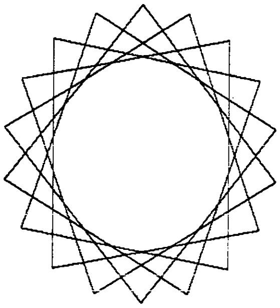
and do these steps over again, I then added we want the computer to wait and suggested that it WAIT 5. The following conversation ensued.



POLY 20 100



POLY 50 100



POLY 100 100

E: Does that mean 5 seconds
C: No
WAIT 60 is a second
E: But if I say 1 that's a second
C: Yes but computers count faster.

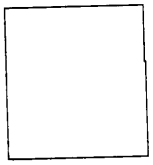
Then Seymour intervened and played a RACE with her. He wrote RACE. It took an input, a starting number and then counted up to 21 by one's. So

RACE→1 2 3 4 5 6 7 8 . . . 21

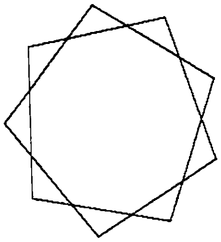
He asked Est to count against the computer. The computer won.

The racing and counting seemed to give another dimension and added more reality to some aspects of the computer:

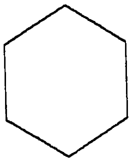
. . . its sequential behavior (After all up to now the results of her programs have been static. So even if



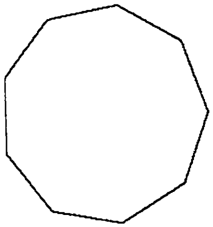
POLY 50 90



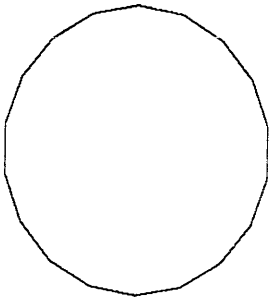
POLY 50 80



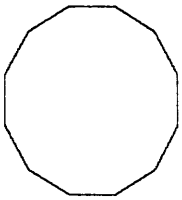
POLY 25 60



POLY 25 40



POLY 20 20



POLY 20 30

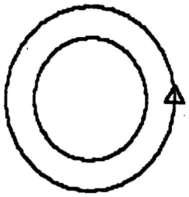
the drawing is sequential, the static may be more real.)

. . . its sense of time

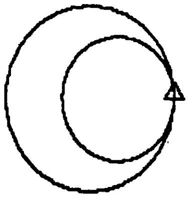
. . . its quickness (even though this isn't in the nano-second range.)

She finished teaching P to the computer. She and I resumed our discussion about recursion. P was to be recursive. We had played the people procedure game (To POW RAISE-ARM LOWER-ARM SAY-POW) last year but she didn't remember. I asked her to be the POW procedure and we worked through it again. Then she made P behave in the same way and ran it. The desired effect was achieved.

I asked her to use LCIRCLE and RCIRCLE to make



she quickly made
out of LCIRCLE 90 and
LCIRCLE 60

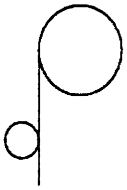


Again an interesting phenomenon (the "start up bug") : she seemed to block until I said "What about moving the turtle?" A trivial piece of advice. Perhaps it really means "stop being complicated, do something simple."

She asked how much to move it. I answered, "Well it walks 90 units to the middle of the big circle and 60 units to the middle of the small so move it 90 take away 60. She did and completed the picture. We then took a break for lunch.

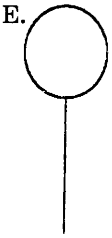
By the way I had tried to accept her first version but she would not give up on the original picture.

After lunch I asked her to make

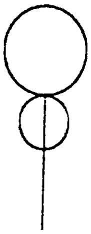


This time using LC or RC (which took diameter as their input).

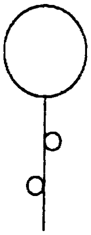
Then I went away. I suspected she had ritualized her experience with the head of RUTH and MARIE. I wanted to see if she could undo it. She worked very hard.



She got



I finally modified the model to

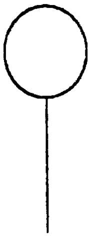


She said no she only wanted 1 leaf. (Esthetics). She did it! She called it DF for dumb flower (again emphatic and dramatic).

In all this I see another interesting phenomenon. Call it making cliches. Are they good or bad? Perhaps necessary. Anyway that's how it seems to go. (Seymour says it now has the blessing of "frame theory" and something cognitive psychologists call "stereotyping". Again I don't know but I'm glad to know that theoretical people are paying attention to the things that seem important. Also, what does Piaget mean by schema?)

Here is a series of events.

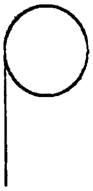
In making the men Est constructed



The obvious first pass at doing this is

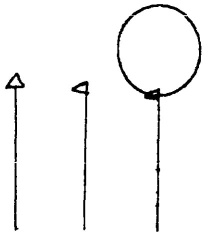
```
FD 50
RCIRCLE 50
```

but this gives

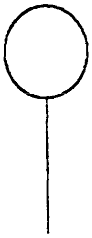


The debugged version is FD 50

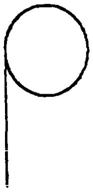
```
LT 90
RCIRCLE 50
```



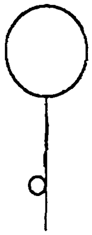
Est worked awhile on this and eventually knew how to make figures like



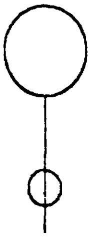
Now the flower model called for



which Est had previously found easier. But now she has trouble. She has "formed a cliche" or "overgeneralized" or whatever. You might say "well she doesn't understand anything. She is echoing mechanically." But it's not so simple. Look at what happened when I suggested putting in a leaf to form:



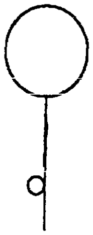
She didn't make



as she would if "LT 90 RCIRCLE" had been completely ritualized. So it's more subtle. Actually Est had developed another cliche. Instead of saying LT 90, she would say RT 90, RT 90, RT 90. In this problem there was payoff. Then she would always turn the turtle right.

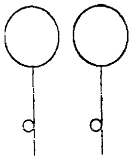
My next suggestion was to make a row of flowers. She did

```
DF
DF
```



but the second DF drew the same flower again. Astonishment! Bug! Again a very "trivial" piece of advice got her going. I pointed to where the next flower could be. No words were necessary (what a lesson for talkative teachers!) . . . she knows how to drive the turtle and she quickly drew a row of flowers.

```
DF
RT 90
FD 50
RT 90
RT 90
RT 90
DF
RT 90
etc.
```



"Fantastic" I said, "and what about one to the left." Serious thinking. It has to go BK 150 she said, "because there are 1 2 3 of them."

Again the research problem: What do these "little" aids mean? What is the learner's problem. This learner sometimes handled this amazingly. Her flower drawer is called DF. Like many children her age she sometimes reverses letters and especially since FD is a LOGO word. So several times her intended DF got the reply:

```
FORWARD NEEDS MORE INPUTS
```

So she wrote a big DF on a piece of paper. Put a circle around it and looked at it ritualistically every time she wanted to draw a flower! (Best model this year of debugging.)

I emphasize: the particular trick for DF was entirely her own idea. If I helped it was by conveying (rather than telling) an attitude to debugging and towards using paper and pencil as a material aid. I had often taken up the pencil in times of difficulty.

Finally another "cliche" which I already mentioned, Est never used LEFT spontaneously. She knew what it did and would oblige if asked to use it. But on her own she would say

RT 90
RT 90
RT 90

rather than

LT 90

There seemed to be no reason to complain or "correct" this perfectly adequate representation! It might be interesting to watch its development. But probably not. One day she will use LT 90 and no one will ever know what happened. Except her, perhaps.

REFERENCES

1. Bamberger, Jeanne, *The Development of Musical Intelligence I: Strategies for Representing Simple Rhythms*, MIT AI Lab. LOGO Memo 19, November 1975.
2. Minsky, Marvin and Seymour Papert, *Artificial Intelligence*, Oregon University Press, 1974. Also as AI Progress Rept., Mass. Inst. Tech., Artificial Intelligence Lab., Memo 252, 1972.
3. Minsky, Marvin, *A Framework for Representing Knowledge*, MIT, A1 Memo 306, 1974.
4. Minsky, Marvin, "Form and Content in Computer Science," *JACM*, Vol. 17, No. 2, 1970. Also as MIT AI Memo 187, 1969.
5. Papert, Seymour and Cynthia Solomon, *Twenty Things to Do with a Computer*, MIT, AI Lab. LOGO Memo 3, July 1971. Also in *Educational Technology*, April 1972.
6. Papert, Seymour, *Teaching Children to be Mathematicians vs. Teaching about Mathematics*, MIT A1 Lab. LOGO Memo 4, July, 1971. Also in Int. J. Math. Educ. Sci. Technol., vol. 3, 249-262, 1972.
7. Papert, Seymour, *Uses of Technology to Enhance Education*, MIT AI Lab. LOGO Memo 8, June 1973.
8. Papert, Seymour, "On Making a Theorem for a Child," *Proc. ACM Annual Conf.* Aug. 1972. Also in *New Educational Technology*, General Turtle Development Inc., Cambridge, Ma.
9. Solomon, Cynthia, "Leading a Child to a Computer Culture," *Proc. ACM SIGCSE-SIGCUE Joint Symposium*, SIGSCE BULLETIN 8,1/SIGCUE TOPICS 2, February 1976. Also as MIT AI LAB LOGO MEMO 20, 1975.

Feature selection for binary data—Medical diagnosis with fuzzy sets*

by JAMES C. BEZDEK
Marquette University
Milwaukee, Wisconsin

ABSTRACT

The notion of fuzzy sets—sets with imprecise boundaries—is a natural cornerstone upon which to build algorithms based on approximate reasoning. Since their inception in 1965 by Zadeh,[1] fuzzy sets have led to first steps towards quantifying data analysis in many fields previously immune to mathematical examination. A fairly exhaustive introduction to the theory and applications of fuzzy sets[2] lists 238 papers dealing with a great variety of recent investigations.

INTRODUCTION

In this paper we discuss the applicability of the fuzzy ISODATA clustering algorithms for (1) dimensionality reduction of binary valued data sets, and (2) computerized medical diagnosis. The first question is often referred to as feature selection; overviews of many popular approaches are available in References 3 and 4. Loosely speaking, one wants to reduce the number of characteristics originally measured to some optimal subset which retains at least as much information about substructure in the data as the original ones. Computerized medical diagnosis is an extremely difficult and ambitious undertaking. Considering the risks involved, enormous improvements need to be made in existing methodologies before the medical community can be asked to rely on the diagnostic suggestions of a computer. However, it is our conviction that the attitude of pessimism displayed in Reference 5 towards this enterprise is largely attributable to the failure of conventional (that is, non-fuzzy) techniques, the results of which must either be accepted at face value or rejected out of hand: we believe that fuzzy sets can be used as a basis for computerized diagnostic *advice* that will provide valuable insight and direction for clinicians with a large data base of previous case histories.

FUZZY CLUSTERING ALGORITHMS

Let R^d denote real, d-dimensional Euclidean space (*feature space*), and let $X=\{x_1,x_2, \ldots , x_n\} \subset R^d$. Each $x_k= (x_{k1},x_{k2}, \ldots , x_{kd}) \in R^d$ is a *feature vector* (subject, patient); each x_{kj} in R is the j^{th} *feature* (characteristic, attribute, symptom) of feature vector x_k; and if every $x_{kj} \in\{0, 1\}$, we call X a binary valued data set. In this instance, we say x_k has attribute j when $x_{kj}=1$, and is lacking it if $x_{kj}=0$. Cluster analysis with respect to X is the problem of finding an integer c, $2\leq c<n$, and c subsets (clusters) of X which partition it into subgroups of points revealing intrinsic substructure in the data. Algorithms to partition X abound; the partitions they find depend on the classification criterion used by the algorithm which defines similarity between pairs of vectors in X.

There are *hard* (i.e., conventional) and fuzzy methods, and each of these main classes can be roughly subdivided into graph-theoretic and objective function techniques. Readers interested in hard algorithms for clustering will find an introduction to the literature in Reference 3; a brief review of fuzzy clustering follows. Clustering with fuzzy sets was first proposed in Reference 6. References 7-10 discuss some of the earliest fuzzy pattern classification schemes. In 1969 Ruspini delineated the first fuzzy clustering method based on objective functions, and foreshadowed the usefulness of information measures (entropy) in the fuzzy sets context. His technique was enlarged and illustrated in References 12-15. Dunn[16] defined the first fuzzy extension of the classical within group sum of squared errors (WGSS) objective functional, and in Reference 17 this approach was generalized to yield the infinite family of algorithms discussed below. Methods of clustering based on fuzzy graphs are still in their infancy; References 18-23 are seminal works in this direction.

CLUSTERING WITH FUZZY ISODATA

A hard c-partition P of X is a collection of c nonempty subsets of X, say $P=\{Y_1,Y_2, \ldots , Y_c\}$, whose

* This research supported by National Science Foundation Grant DCR75-05014.

union is X and whose pairwise intersections are disjoint. To characterize P by a matrix, let $u_i: X \rightarrow \{0, 1\}$ be the characteristic function of Y_i:

$$u_i(x_k) = u_{ik} = \begin{Bmatrix} 1 \text{ in case } x_k \in Y_i \\ 0 \text{ otherwise} \end{Bmatrix}; 1 \leq i \leq c; 1 \leq k \leq n. \tag{1}$$

Denote by V_{cn} the vector space of all real $c \times n$ matrices, let $U \in V_{cn}$, and let u_{ik} be the ik^th entry of U. The set of matrices

$$M_c = \left\{ U \epsilon V_{cn}: u_{ik} \epsilon \{0,1\} \; \forall \; i,k; \sum_{i=1}^{c} u_{ik} = 1 \; \forall \; k; \sum_{k=1}^{n} u_{ik} > 0 \; \forall \; i \right\} \tag{2}$$

is called *hard c-partition space* for X because each partition P of X corresponds uniquely to the matrix in M_c whose rows are the values for the characteristic functions of the subsets in P as shown in Equation (1).

Solutions for all hard clustering algorithms lie in M_c, and this is a fundamental drawback for two reasons: First, each member of the data must be assigned unequivocal membership in one and only one of the c partitioning subsets; however, the substructure in real data rarely—if ever—is so distinct that every member in X is most realistically described as a full member of a single subclass. A fuzzy model can overcome this objection by allowing every individual partial membership in all c subsets, as, for example, one would desire for hybrids when classifying them in parallel with their progenitors. Secondly, M_c is a finite but extremely large set, a complication which often manifests itself in analytical as well as computational intractabilities.

Fuzzy sets provide a natural way to surmount the objections above. We call any *function* u_i that maps X into the closed interval [0,1] a *fuzzy subset* or fuzzy cluster in X. The number $u_i(x_k) = u_{ik}$ is the *grade of membership* of subject x_k in fuzzy set u_i, and fuzzy c-partitions of X are defined by imbedding M_c in

$$M_{fc} = \left\{ U \epsilon V_{cn}: u_{ik} \epsilon [0,1] \; \forall \; i,k; \sum_{i=1}^{c} u_{ik} = 1 \; \forall k; \sum_{k=1}^{n} u_{ik} > 0 \; \forall \; i \right\}. \tag{3}$$

M_{fc} is called *fuzzy c-partition space* associated with X. The requirement that each column in U sum to one stipulates that every vector in X be assigned a *total* membership equal to unity in the partitioning subsets. If M_c is enlarged to include matrices which may have some zero rows, say M_{co}, then M_{fc} is the convex hull of M_{co}.[17] Compactness, convexity, and continuity endow M_{fc} with a pleasant mathematical structure; for example, it has been shown[24] numerically that because M_{fc} is continuous, algorithms defined on it have paths of feasible solutions around undesirable local trap states of algorithms confined to M_c.

Given M_{fc}, how can fuzzy c-partitions of X be found? One way to identify optimal fuzzy clusterings in X is via the family of generalized WGSS error objective functionals defined in Reference 17. On the Cartesian product of M_{fc} with R^{cd}, we define for $m \epsilon [1, \infty)$

$$J_m(U,v) = \sum_{k=1}^{n} \sum_{i=1}^{c} (u_{ik})^m ||x_k - v_i||^2. \tag{4}$$

In (4) $U \epsilon M_{fc}$, $v = (v_1, v_2, \ldots, v_c) \epsilon R^{cd}$, $v_i = (v_{i1}, v_{i2}, \ldots, v_{id}) \epsilon R^d$ for $1 \leq i \leq c$, and $||\cdot||$ is any norm on feature space. J_m is an etxension of the classical minimum variance objective functional J_1 because $J_m = J_1 \; \forall \; m$ whenever $U \epsilon M_c$ is hard. The c vectors $\{v_i\}$ comprising v are presumed to have features prototypical of vectors in X having a high affinity for membership in the respective fuzzy clusters $\{u_i\}$, and so are called *cluster centers* of their respective fuzzy clusters. These vectors will play an important role in the sequel. The measure of similarity in (4) is the norm $||\cdot||$; in this model it compares members of the data to each other indirectly via distances between them and the cluster centers.

Optimal fuzzy c-partitionings of X are defined as part of solution pairs $(\hat{U}, \hat{v})$ of the optimization problem

$$\text{minimize}\{J_m(U,v)\} \quad \text{over } M_{fc} \otimes R^{cd}. \tag{5}$$

Partitions arising as part of solutions for (5) are related to a well defined type of hard, compact, well separated (CWS) clusters for X in Reference 16. There are structured data sets whose clusters do not enjoy this property, but there is a wide class of patterns for which this criterion is very basic, and we adopt it here as implicit in our clustering goals.

Necessary conditions for solutions of (5) were derived[17] for the class of functionals in (4) whose norms were differentiable (e.g., inner product induced norms). It was shown there that for $m > 1$ and $x_k \neq \hat{v}_i \; \forall \; i,k$,

$$\hat{u}_{ik} = \left[\frac{1}{\sum_{j=1}^{c} \left(\frac{||x_k - \hat{v}_i||}{||x_k - \hat{v}_j||} \right)^{\frac{2}{m-1}}} \right], 1 \leq i \leq c; 1 \leq k \leq n, \tag{6a}$$

$$\hat{v}_i = \left[\frac{\sum_{k=1}^{n} (\hat{u}_{ik})^m x_k}{\sum_{k=1}^{n} (\hat{u}_{ik})^m} \right], 1 \leq i \leq c \tag{6b}$$

are necessary in order for $(\hat{U}, \hat{v})$ to be a local solution of (5). Full details for $m = 1$ and the singular cases $x_k = \hat{v}_i$ for some i and k may be found in References 16 and 17. At $m = 1$ requirement (6a) is replaced by a nearest neighbor assignment rule, $U \epsilon M_c$ is necessarily hard, the cluster centers in (6b) are merely the centroids of the hard subsets in U, and the resultant algorithm is essentially the hard ISODATA process of Ball and Hall.[25] For $m > 1$ equations (6) define the

Fuzzy ISODATA algorithms

Choose any $c \times n$ matrix $U_0 \epsilon M_{fc}$. (7a)

Compute the weighted means $\{\hat{v}_i\}$ with U_o and (6b). (7b)

Update $U_o \rightarrow \hat{U}$ with equation (6a). (7c)

Compute the maximum membership defect

$$\max_{i,k}\{|(\hat{U})_{ik} - (U_o)_{ik}|\}. \quad (7d)$$

If less than some prespecified tolerance ϵ, stop. Otherwise relabel $\hat{U} \rightarrow U_o$ and return to (7b).

Implicit in (7) are tie-breaking rules and resolution of singularities. These equations define an iterative optimization procedure for locating approximate minima of J_m. It is convenient to recast this loop in the form of the iterative matrix operator $T_m : M_{fc} \rightarrow M_{fc}$ defined by

$$T_m(\hat{U}_k) = \hat{U}_{k+1} = (T_m)^k(U_o), k=0,1,2, \ldots . \quad (8)$$

Since U's which are part of optimal pairs for J_m must lie among the fixed points of T_m, we call approximate minima of J_m fixed points of fuzzy ISODATA. J_m has the descent property on successive iterates of T_m and the associated set of cluster centers they determine, but it is not now known whether the iterate sequence $\{T_m(\hat{U}_k)\}$ is theoretically convergent. We mention this because the numerical example below suggests an interesting conjecture about these fixed points. The possibility of using T_m to approximate a maximum likelihood operator for certain problems in unsupervised learning is discussed in Reference 26.

SCALAR MEASURES OF PARTITION QUALITY

Since optimal partitionings of X are defined as part of solutions of (5), an obvious way to rank competing partitions is by their corresponding values with J_m. Unfortunately, J_m is not an exception to the fact that global minima of objective functions may suggest very poor interpretations of substructure in X.[24,27,28] Consequently, values of J_m do *not* necessarily rank the merits of different $\hat{U}$'s in M_{fc} as worthwhile clusterings of X. It is here that fuzzy ISODATA departs from conventional clustering techniques, because with hard objective functions the functional values are the only information usually available for addressing this question. With fuzzy partitions however, the fuzziness of $\hat{U}$ allows one to associate various measures of partition quality with $\hat{U}$ which are independent of the method used to produce these partitions. Fuzzy ISODATA is used to generate likely candidates for optimal clusterings of X; their relative quality has been assessed by either of two scalar valued measures defined on M_{fc}:

$F_c(U) = \text{trace}(UU^t)/n$, superscript t being here transpose, (9)

$$H_c(U) = -\left(\sum_{k=1}^{n}\sum_{i=1}^{c} u_{ik} \log_a u_{ik}\right)/n, \text{ with } a\epsilon(1,\infty). \quad (10)$$

$F_c : M_{fc} \rightarrow [1/c, 1]$ was defined in Reference 17 as the *partition coefficient* of U; $H_c : M_{fc} \rightarrow [0, \log_a c]$ was defined in Reference 27 as the *average classification entropy* of U. Although the functional forms of F_c and H_c are quite different, they are related as follows:

$$F_c(U) = 1 \Leftrightarrow H_c(U) = 0 \Leftrightarrow U \epsilon M_c \text{ is hard.} \quad (11a)$$

$$F_c(\tilde{U}) = 1/c \Leftrightarrow H_c(\tilde{U}) = \log_a c \Leftrightarrow \tilde{U} = [1/c]. \quad (11b)$$

$$1 - F_c(U) < \left(\frac{H_c(U)}{\log_a e}\right) < \tfrac{1}{2}(c - F_c(U)). \quad (11c)$$

Equations (11) suggest that the equi-membership partition $\tilde{U} = [1/c]$, i.e., $\tilde{u}_{ik} = 1/c \, \forall \, i,k$, is the fuzziest or worst one can do (geometrically $\tilde{U}$ is the centroid of M_{co}); on the other hand, the ideal situation occurs when the substructure in X is so distinct that a fuzzy algorithm recommends a hard c-partitioning of X. Maximizing F_c over different fixed points of fuzzy algorithms minimizes the total content or overlap in pairwise fuzzy intersections; equivalently, minimizing H_c over the same choices maximizes the "information" extracted from U. In either case, we presume that values of these measures serve as a relative indication of the uncertainty an *algorithm* experiences in trying to assign memberships to the vectors in X. Note that F_c and H_c are well defined for partitions generated by *any* fuzzy clustering method, not just ISODATA; moreover, these functions convey no information about the relative merits of hard c-partitions of X, their usefulness depending entirely on the idea of fuzziness. Numerical evidence indicates that H_c is probably more sensitive than F_c in ranking U's; this has been attributed to the fact that the slope of the logarithmic curve on most of (0,1) is much steeper than that of the parabola (H_c is a sum of logarithmic terms, F_c a sum of parabolic ones). Nonetheless, both measures seem useful, since the lower bound in (11c) is a sharper indication than that in (11a) of how small $H_c(U)$ is.

In general the clustering strategy used with fuzzy ISODATA has been to minimize H_c over approximate fixed points of T_m for whatever alternatives have been considered, and regard the resultant c-partitioning of X as the most optimal one. If this partition is relatively fuzzy (as measured by H_c), we do not infer that X has no well defined substructure; we conclude that none of the algorithms tried have been successful at finding it.

A NUMERICAL EXAMPLE

Table I lists 11 symptoms of 107 stomach disease patients who have either hiatal hernia (patients 1-57) or gallstones (patients 58-107). Table I constitutes our binary data set X. The data was collected as part of a larger study at the Henry Ford Hospital in Detroit by Rinaldo, Scheinok, and Rupe.[30] Various studies utilizing the larger data set for computerized medical

TABLE I—Data Set X: Class 1; Hiatal Hernia

Patient	Symptoms										
1	0	1	0	0	0	1	0	0	0	0	0
2	0	1	0	0	0	1	0	0	0	0	0
3	0	1	0	0	0	1	0	1	0	0	0
4	0	1	0	0	0	1	0	1	0	0	0
5	0	1	0	0	0	1	0	1	0	0	0
6	0	1	0	0	0	1	0	1	0	0	0
7	0	1	0	0	0	1	0	1	1	0	0
8	0	1	0	0	0	1	0	1	1	0	0
9	0	1	0	0	0	1	0	1	1	0	0
10	0	1	0	0	0	1	1	0	0	0	0
11	0	1	0	0	0	1	1	0	1	0	0
12	0	1	0	0	0	1	1	0	1	0	1
13	0	1	0	0	0	1	1	1	0	0	0
14	0	1	0	0	1	1	0	1	0	0	1
15	0	1	0	1	0	1	0	0	1	0	0
16	0	1	0	1	0	1	0	0	1	0	1
17	0	1	0	1	0	1	0	1	0	1	1
18	0	1	0	1	0	1	0	1	1	0	0
19	0	1	0	1	0	1	1	0	1	0	0
20	0	1	0	1	0	1	1	0	1	0	0
21	0	1	0	1	0	1	1	0	1	0	0
22	0	1	0	1	1	0	0	1	1	0	1
23	0	1	1	0	0	0	0	1	1	0	1
24	0	1	1	0	1	0	0	0	0	0	0
25	0	1	1	0	1	0	1	0	0	1	1
26	0	1	1	1	1	0	0	1	0	0	0
27	1	0	0	0	0	0	0	0	0	0	0
28	1	1	0	0	0	0	0	0	0	0	0
29	1	1	0	0	0	0	0	0	1	0	0
30	1	1	0	0	0	0	0	0	1	0	0
31	1	1	0	0	0	0	1	0	0	0	0
32	1	1	0	0	0	0	1	0	1	0	0
33	1	1	0	0	0	1	0	0	1	0	0
34	1	1	0	0	0	1	0	0	1	0	0
35	1	1	0	0	0	1	0	1	0	0	0
36	1	1	0	0	0	1	0	1	0	0	0
37	1	1	0	0	0	1	0	1	1	0	0
38	1	1	0	0	0	1	0	1	1	0	0
39	1	1	0	0	0	1	1	0	0	0	0
40	1	1	0	0	0	1	1	0	0	0	0
41	1	1	0	0	0	1	1	0	1	0	0
42	1	1	0	0	0	1	1	0	1	0	0
43	1	1	0	0	0	1	1	0	1	0	0
44	1	1	0	0	1	0	0	0	1	0	0
45	1	1	0	0	1	0	1	0	0	0	0
46	1	1	0	0	1	0	1	0	0	0	0
47	1	1	0	0	1	0	1	0	1	0	0
48	1	1	0	0	1	1	1	1	1	0	0
49	1	1	0	1	0	0	0	0	1	0	1
50	1	1	0	1	0	1	0	0	0	0	0
51	1	1	0	1	0	1	0	0	0	0	1
52	1	1	0	1	0	1	0	1	0	0	0
53	1	1	0	1	1	0	1	0	0	0	0
54	1	1	0	1	1	0	1	1	1	0	1
55	1	1	1	0	0	1	0	1	1	0	0
56	1	1	1	0	0	1	1	1	1	0	0
57	1	1	1	0	1	1	1	0	0	0	1

TABLE I (Continued)

Patient	Symptoms										
58	0	0	1	0	0	0	0	1	0	1	1
59	0	0	1	0	0	1	0	1	0	0	0
60	0	0	1	0	0	1	0	1	0	0	1
61	0	0	1	0	1	0	0	1	0	1	0
62	0	0	1	1	0	1	0	0	0	0	0
63	0	0	1	1	0	1	0	0	0	0	0
64	0	0	1	1	0	1	0	1	0	0	0
65	0	0	1	1	0	1	0	1	0	0	0
66	0	0	1	1	0	1	0	1	0	0	0
67	0	1	0	0	0	0	1	0	0	0	0
68	0	1	0	0	0	1	0	0	0	0	0
69	0	1	0	0	0	1	0	1	0	0	0
70	0	1	0	0	0	1	0	1	0	0	0
71	0	1	0	0	0	1	1	0	0	0	0
72	0	1	0	0	0	1	1	1	0	0	0
73	0	1	0	0	0	1	1	1	0	0	0
74	0	1	0	1	0	1	0	0	0	0	0
75	0	1	0	1	1	1	1	1	0	0	0
76	0	1	1	0	0	0	0	1	0	0	1
77	0	1	1	0	0	1	0	1	0	0	0
78	0	1	1	0	0	1	0	1	0	0	0
79	0	1	1	0	0	1	0	1	0	1	0
80	0	1	1	0	0	1	0	1	0	1	1
81	0	1	1	1	0	1	0	1	0	0	0
82	0	1	1	1	0	1	0	1	0	0	0
83	0	1	1	1	0	1	0	1	0	0	0
84	0	1	1	1	0	1	0	1	0	0	0
85	0	1	1	1	0	1	0	1	0	0	0
86	0	1	1	1	0	1	0	1	0	0	0
87	0	1	1	1	0	1	0	1	0	0	1
88	0	1	1	1	0	1	0	1	0	1	0
89	0	1	1	1	0	1	1	1	0	0	0
90	1	0	1	0	0	1	0	1	0	0	0
91	1	0	1	1	0	1	0	1	0	0	0
92	1	0	1	1	0	1	0	1	0	0	0
93	1	0	1	1	0	1	0	1	0	0	0
94	1	1	0	0	0	1	0	0	0	0	0
95	1	1	0	0	0	1	0	0	1	0	0
96	1	1	0	0	0	1	0	1	0	0	0
97	1	1	0	0	0	1	0	1	1	0	0
98	1	1	0	0	0	1	0	1	1	1	0
99	1	1	0	0	1	1	0	1	1	0	0
100	1	1	0	1	0	1	0	0	0	0	0
101	1	1	1	0	0	1	0	0	0	0	1
102	1	1	1	0	0	1	0	1	0	0	0
103	1	1	1	0	0	1	0	1	0	0	0
104	1	1	1	0	0	1	0	1	0	0	1
105	1	1	1	0	0	1	1	0	0	0	0
106	1	1	1	1	0	1	0	0	0	0	0
107	1	1	1	1	0	1	0	0	0	0	0

diagnosis have been discussed in References 31-35; full details on the data are available in Reference 31. The 11 symptoms measured (present=1, absent=0) were:

Symptom	*Description or type of Abdominal Pain*
1	Male=1; Female=0
2	Epigastric Pain
3	Upper right quadrant pain
4	Back pain
5	Discomfort episodes of 1-4 weeks
6	Discomfort episodes of 0-1 days
7	Relief induced by food ingestion
8	Aggravation induced by food ingestion
9	Aggravation induced by position
10	Weight loss (at least 20 lbs. in 6 mos.)
11	Persistence (at least 1 month in length)

The calculations were made in single precision Fortrans IV using logarithms in (10) to base e=2.718. . . . The convergence threshold ϵ used in (7b) was $\epsilon=0.01$. Because fuzzy ISODATA—like all hill climbing methods—is susceptible to stagnation at local minima of J_m, it is necessary to test the stability of fixed points of T_m by varying the initial guess for U_o in (7a). Other studies report results concerning this parameter; for this example the only initial guess used is

$U_o=\alpha U+\beta\tilde{U}$, where $\alpha=\sqrt{1/2}$, $\beta=1-\sqrt{1/2}$, and

$$U=\left[\underbrace{\begin{matrix}1\,0\,.\,.\,.\,0\\0\,1\,.\,.\,.\,0\\ \vdots \qquad \vdots\\ 0\,0\,.\,.\,.\,1\end{matrix}}_{c\times c}\;\middle|\;\underbrace{\begin{matrix}1\,1\,.\,.\,.\,1\\0\,0\,.\,.\,.\,0\\ \vdots\qquad\vdots\\0\,0\,.\,.\,.\,0\end{matrix}}_{c\times(n-c)}\right] \tag{12}$$

U_o is an initial guess lying midway between $\tilde{U}$ and the hard c-partitions of X, as measured by the value of F_c, since $F_c(U_o)=1/2+(1/2c)$. Only one initial guess is used in this example to shorten the presentation of numerical results.

Finding c is often the most important and difficult problem in clustering. The use of fuzzy ISODATA for this purpose is discussed elsewhere; in this investigation we fix c=2 in the interests of brevity. The algorithmic parameters varied here are the weighting exponent m and norm $||\cdot||$ appearing in (4). Values for m are 1.10, 1.33, 1.67, and 2.00. Results with other values are contained in Reference 29. Three norms induced by the weighted inner product $\langle x,x\rangle=x^tAx$ on R^d were used. These norms were realized by different choices for the symmetric matrix A:

N1 (Euclidean) induced by A=I, the d×d identity. (13a)

N2 (Diagonal) induced by $A=[\text{diag}(\sigma_1^2,\ldots,\sigma_d^2)]^{-1}$, the inverse of the diagonal matrix of marginal sample feature variances. (13b)

N3 (Mahalonobis) induced by $A=[\text{cov}(X)]^{-1}$, the inverse of the sample covariance matrix. (13c)

Further discussion on these choices may be found in Reference 35. Having established the computing protocols, we turn to the numeric results.

Table II lists entropies H_2 and their lower bounds $1-F_2$ for the fixed points of T_m obtained by processing X with (7) under the assumptions above. These values are comparable only for fixed values for m, because as m→1, partitions obtained by fuzzy ISODATA are always "less fuzzy" in the sense of F_c or H_c. Since (7) represents an infinite family of algorithms, there is the practical question of which one to use. The only theoretical result concerning this to date appears in Reference 36, where an analogy to minimum resistance electrical networks is used to suggest that only J_2 extends the physical interpretation of J_1 made there. It will be seen in Table II that ISODATA proceeds from U_o to $\tilde{U}$ (quite rapidly) for every norm at m=2.00; for N2 and N3 at m=1.67; and for N3 at m=1.33 (recall from (11b) that with c=2, F_2=.500 if and only if evaluated at $\tilde{U}$). Whether or not other initial guesses for U_o would lead to this fixed point is a matter of speculation; the rather surprising conjecture suggested by this observation is that the size of stability domains of fixed points of T_m is dependent on both m and the norm in (4). Of course, as m→∞, $\tilde{U}$ becomes the *only* fixed point of T_m, as is evident from (6a). Table II shows that it may be necessary to experimentally decrease m towards m=1 until fuzzy ISODATA successfully begins to avoid equi-memberships for a given set of data.

The values in Table II also indicate a slight preference for the Euclidean norm over N2, and a definite preference compared to N3, so we infer that this data

TABLE II—Entropies for Data Set X

Weighting Exponent m	Norm $\|\cdot\|$	Lower Bound $1-F_2(\hat{U})$	Entropy $H_2(\hat{U})$
1.10	N1	.051	.088
	N2	.057	.095
	N3	.086	.162
1.33	N1	.253	.397
	N2	.274	.425
	N3	.500	.693
1.67	N1	.420	.608
	N2	.500	.693
	N3	.500	.693
2.00	N1	.500	.693
	N2	.500	.693
	N3	.500	.693

is most separable by N1 among the three norms considered. Accordingly, the norm in (4) for subsequent runs is now fixed at $||\cdot||=N1$, and in view of the results obtained at $m=1.67$ and 2.00, we drop these values for the weighting exponent.

In Table III are listed the membership functions corresponding to terminal partitions obtained with (7), $||\cdot||=N1$, and $m=1.10$, 1.33. Scanning these values, one quickly obtains a feel for which members of the data indicate a strong desire to be classified into one subclass or the other. As we expect, the partition of X associated with the smaller value of m is very nearly hard, while the second partition begins to exhibit clearly those subjects in X apparently causing the most difficulty to ISODATA in assigning memberships. This feature of fuzziness—*identification of the troublesome or distinguished individuals in the data*—is perhaps the most important reason for using fuzzy models. Information of this kind is simply not available when using hard classification procedures, for then all the entries in solution partitions are 0's and 1's.

Since any discussion of error rates presumes a comparison with hard labels, it is necessary to convert fuzzy partitions into hard ones before this is possible. An obvious (but not necessarily best) way to do this is via the maximum membership rule: assign each $x_k \epsilon X$ to the cluster in which it holds maximum membership. All error rates mentioned below are computed with hard 2-partitions of the data obtained in this fashion. With this convention in mind, we have from Table III at $m=1.10$ 23 incorrect labels, and at $m=1.33$, 25 mislabelled patients. We emphasize that these are *not* classifier performance rates, because we are clustering here; no attempt is being made to train a classifier for prediction with unlabelled samples. However, we presume these figures are indicative of error rates which may be obtained with fuzzy classifiers now under study. Of more immediate interest is the way we can use fuzzy ISODATA to attack the feature selection problem.

FEATURE SELECTION USING FUZZY ISODATA

Contrary to one's intuition, adding more features does not always lead to better classifier performance.[3] In some instances the converse is true; deletion of features may remove the source of confusion preventing an algorithm from detecting substructure known (or presumed) to exist in the data, and in any event, reduction of the dimension of feature space alleviates the computational burden imposed by using many features. In medicine, this amounts to asking for the minimum number of symptoms needed to detect a particular disease, or to discriminate between closely related ones. The basis for a technique of fuzzy feature selection using algorithm (7) is contained in the simple

Proposition Let X be any binary valued data set, $X=\{x_1, \ldots, x_n\}$ contained in R^d; let $\{\hat{v}_i\}$ be the cluster centers given by (6b); and suppose $\hat{v}_{ij}$ to be the j^{th} component of $\hat{v}_i$ for $1\leq i\leq c$; $1\leq j\leq d$. Then

$$0\leq \hat{v}_{ij}\leq 1 \ \forall\ i,j \tag{14a}$$

$$\hat{v}_{ij}=0 \Rightarrow x_{1j}=x_{2j}=\ldots=x_{nj}=0 \tag{14b}$$

$$\hat{v}_{ij}=1 \Rightarrow x_{ij}=x_{2j}=\ldots x_{nj}=1 \tag{14c}$$

Proof Rewriting equation (6a) in the form $\hat{u}_{ik}=(1/1+c_{ik})$, where c_{ik} is the sum in the denominator over $j=1,2,\ldots, c$ with $j\neq i$, we observe that $c_{ik}>0$ for every i and k, hence $\hat{u}_{ik} \epsilon (0,1)$ for every i and k. In view of this the denominator in (6b), $\sum_{k=1}^{n} (\hat{u}_{ik})^m>0$ for $1\leq i\leq c$. Now consider the component form of (6b) for any j;

$$\hat{v}_{ij}=\sum_{k=1}^{n}\left[\frac{(\hat{u}_{ik})^m}{\sum_{s=1}^{n}(\hat{u}_{is})^m}\right]x_{kj};\ 1\leq j\leq d;\ 1\leq i\leq c. \tag{15}$$

The coefficients of x_{kj} in (15) are all strictly positive, and since every x_{kj} is greater than or equal to zero, $\hat{v}_{ij}$ is also. Moreover, this also shows that $\hat{v}_{ij}$ can equal zero if and only if all the x_{kj} in (15) are zero. Finally, since every x_{kj} in (15) is less than or equal to one, that sum is bounded above by 1, the number obtained upon replacing all of the x_{kj}'s with 1. Since the maximum is attained when this occurs, the proof is complete.

We elaborate the implications of equations (14) for feature selection by the following series of observations:

(i) $\hat{v}_{ij}=0$: Since the proof is independent of i, it's easy to see that an even stronger statement holds: $\hat{v}_{ij}=0 \Leftrightarrow \hat{v}_{kj}=0$ for $1\leq k\leq c$ with $k\neq i$. From (14b) it follows that this occurs when and only when attribute j is absent from all n members of the data, in which case it is irrelevant to substructure in X (medically, no patient had symptom j).

(ii) $\hat{v}_{ij}=1$: As in (i), the stronger statement $\hat{v}_{ij}=1$ if and only if all the $\hat{v}_{kj}$'s with $k\neq i$ are 1 holds. In this event, feature j is a maximal descriptor of the n individuals in X (medically, all patients had symptom j).

(iii) $0<\hat{v}_{ij}<1$: Again, this can happen when and only when $0<\hat{v}_{kj}<1$ for all $k\neq i$. Ostensibly, feature j has a variable amount of influence in describing members of the c subgroups in X. This suggests that the relative magnitudes of $\hat{v}_{1j}, \hat{v}_{2j}, \ldots, \hat{v}_{cj}$ may rank the efficacy of j as a descriptor of each subclass (medically, some patients in each subclass had symptom j, and others did not).

(iv) Combining (i)-(iii), it is seen that one of the cluster centers $\hat{v}_i$ is *entirely* binary valued if and only if *all c of them are,* and this occurs if and only if all n members of the data are identical. In this eventuality there is no possibility for mathematical (or medical) detection of subclasses in X. On the other hand, we

TABLE III—Membership Functions Obtained by Fuzzy ISODATA

Patient	m=1.10		m=1.33	
	$\hat{u}_2$	$\hat{u}_1$	$\hat{u}_2$	$\hat{u}_1$
1	.001	.999	.280	.720
2	.001	.999	.280	.720
3	.137	.863	.833	.167
4	.137	.863	.833	.167
5	.137	.863	.833	.167
6	.137	.863	.833	.167
7	.002	.998	.420	.580
8	.002	.998	.420	.580
9	.002	.998	.420	.580
10	.000	1.000	.118	.882
11	.000	1.000	.048	.952
12	.000	1.000	.114	.886
13	.005	.995	.464	.536
14	.389	.611	.671	.329
15	.002	.998	.252	.748
16	.022	.978	.341	.659
17	.959	.041	.820	.180
18	.092	.908	.638	.362
19	.000	1.000	.137	.863
20	.000	1.000	.137	.863
21	.000	1.000	.137	.863
22	.233	.767	.499	.501
23	.917	.083	.685	.315
24	.735	.265	.554	.446
25	.558	.442	.466	.534
26	.999	.001	.867	.133
27	.015	.985	.230	.770
28	.000	1.000	.086	.914
29	,000	1.000	.038	.962
30	.000	1.000	.038	.962
31	.000	1.000	.049	.951
32	.000	1.000	.025	.975
33	.000	1.000	.022	.978
34	.000	1.000	.022	.978
35	.003	.997	.435	.565
36	.003	.997	.435	.565
37	.000	1.000	.152	.848
38	.000	1.000	.152	.848
39	.000	1.000	.032	.968
40	.000	1.000	.032	.968
41	.000	1.000	.013	.987
42	.000	1.000	.013	.987
43	.000	1.000	.013	.987
44	.000	1.000	.068	.932
45	.000	1.000	.079	.921
46	.000	1.000	.079	.921
47	.000	1.000	.046	.954
48	.000	1.000	.113	.887
49	.006	.994	.176	.824
50	.003	.997	.239	.761
51	.038	.962	.340	.660
52	.236	.764	.684	.316
53	.004	.996	.153	.847

TABLE III (Continued)

Patient	m=1.10		m=1.33	
	$\hat{u}_2$	$\hat{u}_1$	$\hat{u}_2$	$\hat{u}_1$
54	.020	.980	.258	.742
55	.507	.493	.576	.424
56	.052	.948	.331	.669
57	.149	.851	.324	.676
58	.998	.002	.834	.166
59	1.000	.000	.968	.032
60	1.000	.000	.936	.064
61	.994	.006	.796	.204
62	1.000	.000	.884	.116
63	1.000	.000	.884	.116
64	1.000	.000	.978	.022
65	1.000	.000	.978	.022
66	1.000	.000	.978	.022
67	.000	1.000	.126	.874
68	.001	.999	.280	.720
69	.137	.863	.833	.167
70	.137	.863	.833	.167
71	.000	1.000	.118	.882
72	.005	.995	.464	.536
73	.005	.995	.464	.536
74	.059	.941	.546	.454
75	.155	.845	.576	.424
76	.998	.002	.865	.135
77	1.000	.000	.987	.013
78	1.000	.000	.987	.013
79	1.000	.000	.940	.060
80	.999	.001	.904	.096
81	1.000	.000	.992	.008
82	1.000	.000	.992	.008
83	1.000	.000	.992	.008
84	1.000	.000	.992	.008
85	1.000	.000	.992	.008
86	1.000	.000	.992	.008
87	1.000	.000	.967	.033
88	1.000	.000	.960	.040
89	.999	.001	.909	.091
90	.999	.001	.883	.117
91	1.000	.000	.923	.077
92	1.000	.000	.923	.077
93	1.000	.000	.923	.077
94	.000	1.000	.068	.932
95	.000	1.000	.022	.978
96	.003	.997	.435	.565
97	.000	1.000	.152	.848
98	.005	.995	.283	.717
99	.001	.999	.191	.809
100	.003	.997	.239	.761
101	.731	.269	.525	.475
102	.997	.003	.888	.112
103	.997	.003	.888	.112
104	.995	.005	.836	.164
105	.034	.966	.239	.761
106	.958	.042	.674	.326
107	.958	.042	.674	.326

find that if and only if a single cluster center has *no* component either 0 or 1, then all c cluster centers are of this type.

In view of these remarks, it seems natural to call the components $\{\hat{v}_{ij}\}$ of cluster center $\hat{v}_i$ the *feature centers* of class i. Table IV exhibits values of these centers for each of the fuzzy partitions listed in Table III. The ranking of symptom importance for patients with hiatal hernia (class 1) established by values of $\{\hat{v}_{ij}\}$ at either value of m is $2>6>1>9 \ldots >10$. We infer from this that among the 11 symptoms measured, epigastric pain (2) is most likely to occur in patients with this disorder, whereas they will exhibit weight loss (10) only occasionally. To see whether the magnitudes of the $\hat{v}_{ij}$'s really do this, let p_{ij} be the relative frequency of occurrence of symptom j in class i patients. From Table I we find that $p_{12}=0.982$, $p_{1,10}=0.035$. These frequencies should be compared to the values of the corresponding feature centers for class 1: for example, with $m=1.10$ we have $\hat{v}_{12}=0.985$, and $\hat{v}_{1,10}=0.021$. These comparisons seem to corroborate our supposition concerning the ability of the fuzzy feature centers to rank the significance of the features as descriptors of each class.

For patients with gallstones (class 2), there is some shifting in ranks established by changing m from 1.10 to 1.33; this seem to indicate that members of this class are somewhat less distinctive. Nonetheless, we find from Table IV that in both cases, the most important features are {3,6,8,2}; the least important are {5,7,9}. Of course, one may take the opposite view, and regard {5,7,9} as the features most important for deciding a patient does *not* have gallstones. This remark points up the fact that the $\hat{v}_{ij}$'s do not establish which features possess discriminatory power for separating class i from closely related classes, and at the same time, suggests a way to use the feature centers for *pairs* of subclasses to select optimal discriminators.

TABLE IV—Cluster Centers for the Membership Functions in Table III

Exponent	Symptom	Feature Centers		Absolute
		(Hernia)	(Galls.)	Differences
m	j	$\hat{v}_{1j}$	$\hat{v}_{2j}$	$\lvert\hat{v}_{1j}-\hat{v}_{2j}\rvert$
1.10	1	.570	.269	.302
	2	.985	.668	.317
	3	.063	.929	.865
	4	.226	.551	.324
	5	.174	.104	.070
	6	.770	.837	.068
	7	.418	.048	.370
	8	.393	.844	.451
	9	.479	.044	.435
	10	.021	.165	.144
	11	.117	.251	.134
1.33	1	.654	.260	.394
	2	.974	.752	.222
	3	.105	.686	.581
	4	.214	.485	.271
	5	.191	.098	.093
	6	.713	.878	.164
	7	.467	.092	.375
	8	.285	.839	.553
	9	.527	.103	.423
	10	.031	.118	.087
	11	.127	.198	.071

An obvious indication of "how separable" classes i and j are is their cluster center separation $||\hat{v}_i-\hat{v}_j||$. This measure, however, suppresses the information we want to use for reducing the number of features required to effect the classification. A more suitable measure is afforded by the vector of absolute differences of the components of $\hat{v}_i$ and $\hat{v}_j$: for all i and j let

$$f_{ij}=(|\hat{v}_{i1}-\hat{v}_{j1}|,|\hat{v}_{i2}-\hat{v}_{j2}|,\ldots,|\hat{v}_{id}-\hat{v}_{jd}|). \quad (16)$$

The components of $f_{ij,k}$ of vector f_{ij} measure feature center separations between the feature centers for classes i and j. Equations (14) lead to the following results for these components:

$$0\leq f_{ij,k}\leq 1 \text{ for all i,j, and k.} \quad (17a)$$

$f_{ij,k}=0 \Leftrightarrow$ Either all or none of the vectors in both classes i and j have feature k. (17b)

$f_{ij,k}=1 \Leftrightarrow$ All vectors in class i and no vectors in class j have feature k, or vice versa. (17c)

We presume feature k to be either useless or optimal as a discriminator between classes i and j according as (17b) or (17c) respectively occurs. (17a) shows these to be the extremes, intimating that the values $f_{ij,1}$, $f_{ij,2}, \ldots, f_{ij,d}$ rank by their magnitudes the relative utility of the d features for discrimination between classes i and j.

To test this speculation, the vector f_{12} defined by (16) corresponding to the cluster centers in Table IV was used to identify the optimal feature subsets of dimensions 1,2, and 3, and the data set X was reprocessed with fuzzy ISODATA using only these features. The last column of Table IV reports the values of $f_{12,k}$: evidently symptom 3—upper right quadrant pain—is implicated as the most powerful attribute for distinguishing between gallstones and hiatal hernia. The feature center values $\hat{v}_{13}=0.105$ and $v_{23}=0.686$ suggest that very few hernia patients suffer from symptom 3, while most gallstones patients may be expected to have it. Indeed, from Table I we find that the relative frequencies of symptom 3 are $p_{13}=0.123$ and $p_{23}=0.680$ respectively. Continuing in this fashion, we deduce that either {3,8} or {3,9} would be the best 2-dimensional subset of features to use; that {3,8,9} is the best set of 3 features at either value of m; and so on.

The results of clustering these feature subsets are reported in Table V as numbers of misclassifications stemming from the hard 2-partitions realized by maximum membership conversion of the associated fuzzy fixed points of T_m. Using symptom 3 alone results in exactly the same hard partitions as using symptoms 3 and 9; moreover, it will be seen that the overall error rates achieved with either of these subsets is at least

TABLE V—Misclassifications * Using Reduced Feature Spaces

Symptoms	m=1.10			m=1.33		
	Galls.	Hernia	Overall	Galls.	Hernia	Overall
Used	n_1=50	n_2=57	n=107	n_1=50	n_2=57	n=107
1-11	17	6	23	13	12	25
3	16	7	23	16	7	23
3,9	16	7	23	16	7	23
3,8	13	23	36	13	23	36
3,8,9	13	23	36	10	17	27
Deleted#	n_1=43	n_2=41	n=84	n_1=43	n_2=41	n=84
1-11	0	7	7	0	7	7

*Based on hard (maximum membership) partitions.

#Data X with patients {23-26,55-57,67-75,94-100} deleted.

as good as the rate attained using all 11 features. Note that symptoms 3 and 9 are much less effective than 3 and 8, and the error rate using {3,8,9} is in between the best and worst ones shown. From these results it appears that the feature selection method proposed above successfully extracts a small number of features which possess essentially the same information relevant to substructure in X detected by fuzzy ISODATA as the original ones.

SUMMARY

Fuzzy clustering, and in particular fuzzy ISODATA, has been reviewed, and is proposed here as a basis for a new technique applicable to the problem of feature selection. Specifically, equations (14) and (17) seem useful in ranking the effectiveness of binary valued features both as subclass representatives and as discriminators between pairs of fuzzy subclasses in X. A numerical example was presented which seems successful enough to warrant further investigations into the plausibility of the method. We note that this technique is applicable *only* for binary data sets: in fact, (6b) shows that the cluster centers $\{\hat{v}_i\}$ lie in the linear subspace generated by the data, so when the features have continuous domains, (14) and (17) are invalid.

As a means of computerized medical diagnosis, the technique described above is incomplete in the sense that it is a clustering method, not a classifier. Nonetheless, it seems fair to assert that our example exhibits the promise fuzzy sets may hold for this problem. Our conviction is that fuzziness is the premise needed as a basis for pattern recognition; more precisely, we think it an appropriate generalization of the conventional strategies criticized in Reference 5. The reason for this lies with the fuzzy membership values generated by algorithms like fuzzy ISODATA; not only do they indicate a patient's relative affinity for having every disease represented by members of the data; but perhaps more importantly, low memberships can be used to identify those patients whose symptoms indicate further personal attention. For example, the values in Table III suggest that the 23 patients whose Table I labels are {23-26,55-57,67-75,94-100} are—by virtue of their relatively low memberships—the ones most affecting the computer's success at separating the two subclasses. If these 23 individuals are deleted from X, and the remaining 84 patients are processed with ISODATA, the results reported in the last row of Table V indicate an increase of about 14 percent in the accuracy of labelling obtained on all 11 features with either value of m. This appears to confirm that low memberships signal troublesome patients. (Note that processing this deleted set with only feature 3 results in a recognition rate of 100 percent: the 23 patients identified above are precisely the 23 subjects having the "uncharacteristic" labels for members of their classes with respect to feature 3 alone). Of course, it is not the business of the medical community to *delete* troublesome patients from data sets for the convenience of a computer: on the contrary, these are the patients that doctors want most to *identify*, and we believe that fuzzy methodologies will eventually be useful in realizing computer assistance and counseling for people in this profession.

REFERENCES

1. Zadeh, L., "Fuzzy Sets," *Inf. and Control,* 8, 1965, pp. 338-353.
2. Zadeh, L., K. Fu, K. Tanaka and M. Shimura, *Fuzzy Sets and Their Applications to Cognitive and Decision Processes,* Academic Press, New York, 1975.
3. Duda, R. and P. Hart, *Pattern Classification and Scene Analysis,* Wiley-Interscience, New York, 1973.
4. Tou, J. and R. Gonzalez, *Pattern Recognition Principles,* Addison Wesley, Reading, 1975.
5. Croft, D., "Mathematical Methods in Medical Diagnosis," *Annals Bio. Engr.,* 2, 1974, pp. 68-89.
6. Bellman, R., R. Kalaba, and L. Zadeh, "Abstraction and Pattern Classification," *Jo. Math. Anal. and Appl.,* 13, 1966, pp. 1-7.
7. Chang, C., *Fuzzy Sets and Pattern Recognition,* PhD Thesis, Univ. of California at Berkeley, 1967.
8. Wee, W., *On a Generalization of Adaptive Algorithms and Applications of the Fuzzy Set Concept to Pattern Classification,* Tech. Rep. 67-7, EE Dept., Purdue Univ., Lafayette, Indiana, 1967.
9. Flake, R. and B. Turner, "Numerical Classification for Taxonomic Problems," *Jo. Theo. Bio.,* 20, 1968, pp. 260-270.
10. Gitman, I. and M. Levine, "An Algorithm for Detecting Unimodel Fuzzy Sets and Its Application as a Clustering Technique," *IEEE Trans. Comp.,* C-19, 1970, pp. 917-923.
11. Ruspini, E., "A New Approach to Clustering," *Inf. and Control,* 15, 1969, pp. 22-32.
12. ———, "Numerical Methods for Fuzzy Clustering," *Inf. Sciences,* 2, 1970, pp. 319-350.
13. ———, *New Experimental Results in Fuzzy Clustering,* Internal Report, Brain Research Institute, UCLA, 1970.

14. ———, *Applications of Fuzzy Clustering to Pattern Recognition*, Internal Report, Brain Research Institute, UCLA, 1970.
15. Larsen, L., E. Ruspini, J. McNew, D. Walter and W. Adey, "A Test of Sleep Staging Systems in the Unrestrained Chimpanzee," *Brain Research*, 40, 1972, pp. 319-343.
16. Dunn, J., "A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters," *Jo. Cybernetics*, 3,3, 1974, pp. 32-57.
17. Bezdek, J., *Fuzzy Mathematics in Pattern Classification*, PhD Thesis, Applied Mathematics, Cornell Univ. Ithaca, 1973.
18. Zadeh, L., "Similarity Relations and Fuzzy Orderings," *Inf. Sciences*, 3, 1971, pp. 177-200.
19. Tamura, S., S. Niguchi and K. Tanaka, "Pattern Classification Based on Fuzzy Relations," *IEEE Trans.* SMC, SMC-1, 1971, pp. 61-66.
20. Dunn, J., "A Graph-Theoretic Analysis of Pattern Classification via Tamura's Fuzzy Relation," *IEEE Trans.* SMC, SMC-4, 1974, pp. 310-313.
21. Kandel, A. and L. Yelowitz, "Fuzzy Chains," *IEEE Trans.* SMC, SMC-4, pp. 472-475.
22. Yeh, R. and S. Bang, *Fuzzy Relations, Fuzzy Graphs, and their Applications to Clustering Analysis*, CS Report SESLTC-3, Univ. of Texas at Austin, 1974.
23. Rosenfeld, A., "Fuzzy Graphs," in *Fuzzy Sets and Their Applications to Cognitive and Decision Processes*, Zadeh et al., eds., Academic Press, New York, 1975.
24. Bezdek, J., "Cluster Validity with Fuzzy Sets," *Jo. Cybernetics*," 3,3, 1974, pp. 58-73.
25. Ball, G. and D. Hall, "A Clustering Technique for Summarizing Multivariate Data," *Behav. Sci.*, 12, 1967, pp. 153-155.
26. Bezdek, J. and J. Dunn, "Optimal Fuzzy Partitions: A Heuristic for Estimating the Parameters in a Mixture of Normal Distributions," *IEEE Trans. Comp.*, Aug., 1975, pp. 835-838.
27. Wishart, D., "Mode Analysis: A Generalization of Nearest Neighbor Which Reduces Chaining Effects," in *Numerical Taxonomy*, Cole, A. ed., Academic Press, New York, 1969, pp. 282-308.
28. Ling, R., *Cluster Analysis*, PhD Thesis, Yale Univ., New Haven, 1971.
29. Bezdek, J., "Mathematical Models for Systematics and Taxonomy," in *Proc. Eighth Int. Conf. on Numerical Taxonomy*, G. Estabrook, Ed., Freeman, San Francisco, 1975.
30. Rinaldo, J., P. Scheinok, and C. Rupe, "Symptom Diagnosis: A Mathematical Analysis of Epigastric Pain," *Ann. Int. Medicine*, 59, 1963, pp. 145-154.
31. Scheinok, P. and J. Rinaldo, "Symptom Diagnosis: Optimal Subsets for Upper Abdominal Pain," *Comp. and Bio. Res.*, 1, 1967, pp. 221-236.
32. Scheinok, P., "Symptom Diagnosis: Bayes' Theorem and Bahadur's Distribution," *Bio-Med. Comp.*, 3,1, 1973, pp. 17-28.
33. Cumberbatch, J. and H. Heaps, "Application of a Non-Bayesian Approach to Computer Aided Diagnosis for Upper Abdominal Pain," *Bio-Med. Comp.*, 1973.
34. Toussaint, G. and P. Sharpe, "An Efficient Method for Estimating the Probability of Misclassification Applied to a Problem in Medical Diagnosis," *Comp. Bio. Med.*, 4, 1975, pp. 269-278.
35. Bezdek, J. "Numerical Taxonomy with Fuzzy Sets," *Jo. Math. Bio.*, 1,1, 1973, pp. 57-71.
36. ———, "A Physical Interpretation of Fuzzy ISODATA," *IEEE Trans.* SMC, in press.

Procedural representation in a fuzzy problem-solving system

by RICHARD A. LeFAIVRE
Rutgers University
New Brunswick, New Jersey

ABSTRACT

This paper addresses the problem of developing new representational structures for use in problem-solving domains which are imprecise or uncertain in nature. The incompatibility of precise representations with complex systems is discussed, and an argument is made that the inability to arrive at symbolic representations, not any inherent "complexity," is what makes natural reasoning difficult to model. The programming language FUZZY, which provides a number of facilities for representing and manipulating fuzzy information, is described briefly, and the use of "procedure demons" to specify global control regimes is examined. The procedural mechanisms available in FUZZY are then used to analyze a simple problem of fuzzy deductive inference.

INTRODUCTION

The ability to effectively utilize information which is vague, imprecise, or uncertain is of inestimable importance to a problem-solving system which is confronted with a world view even approaching the "real world" in complexity. It is clear that human problem-solvers are able to function in inexact domains long before they can manipulate the abstract symbols we create to represent "exact" reasoning (e.g., consider how you were able to recognize your mother as a two-year-old). But it is precisely the kind of vague, imprecise reasoning which humans seem to do so well which perplexes researchers in the developing field of cognitive science* who are concerned with understanding and modeling the thought process. Relatively well-defined problem domains such as game playing and theorem proving have been most amenable to analysis and successful simulation. Problem areas which are somewhat more imprecise—that is, whose "rules" and solution algorithms are vaguely specified—have proven to be more difficult to model at a level approaching that of human performance (consider simple scene description and speech understanding). And other domains even less amenable to precise analysis have yet to be even considered by researchers in artificial intelligence, although they constitute the bulk of the average human's reasoning effort (how does one judge that a particular face is "pretty"; decide whether or not to get married, and to whom; figure out whether to go to a play or a movie tonight, and, if so, which one?).

The problem of representation

Zadeh[24,25] describes what he calls the *principle of incompatibility,* which states that the complexity of a given domain is more or less inversely related to the precision (exactness) with which it can be analyzed. Although this tenant seems to hold true in many situations, and can indeed be made a truism by utilizing it as a definition of the ill-defined term "complexity," it is somewhat unsatisfying. Do we really believe that the ability to recognize a chair as a chair and not a table, bench, footstool, etc., is a more complex action than proving a theorem in predicate calculus or playing a reasonable game of chess, simply because we can analyze the latter more precisely than the former? It appears to me that the problem lies not in the underlying complexity of the task, but in the level at which we can (symbolically) *represent* our knowledge about the particular task and its problem domain. It happens that we have developed precise mechanisms for representing our knowledge about the complex reasoning utilized in game playing and theorem proving (involving constraints, rules of action, searching techniques, etc.), whereas the simple task of recognizing a chair (much less a pretty face) has defied attempts at symbolic representation. The reason, of course, is that games and formal logics exist in artificial domains which are *of necessity* described symbolically. Dealing with the real world involves a *natural* form of reasoning which must be developed well before artificial symbolic reasoning appears, and the resulting mechanisms involved in such natural reasoning are correspondingly harder to introspect about (and, therefore, to translate into symbolic representations).

Unfortunately, the only mechanisms we have available for artificially representing knowledge are sym-

* We are indebted to Bobrow and Collins[3] for the popularization of this term.

bolic in nature—a situation which will in all likelihood remain unchanged for some time to come. Our task is thus to develop more general and powerful representational structures, along with techniques for integrating and utilizing these structures to their best advantage. At the same time we must search for ways of representing so-called "real-world" (i.e., naturally non-symbolic) knowledge in such a way that it can be effectively utilized in the problem-solving process. It is this problem that the research reported here addresses.

Previous research

Past investigations into the problem of representing knowledge which is vague, imprecise, uncertain, inexact, or probabilistic in nature can be grouped into four major classifications:

(1) Philosophical and logical foundations—Philosophers of language have long contended that any symbolic system (i.e., language) used to describe the real world must, of necessity, admit concepts which are ill-defined. Indeed, it can be argued that it is precisely the fuzziness of natural language (and the underlying cognitive structure upon which it is built) that allows us to communicate in a meaningful way. The infinite complexity of our world makes it necessary for us to summarize, generalize, compress, and otherwise eliminate the mass of irrelevant detail with which we are continually confronted. Formal logicians, who are concerned with representing knowledge in a formal system, discovered quite early that classical logic with its true-false dichotomy was not sufficient to represent the full range of human knowledge. Various attempts were thus devised to liberalize this dichotomy, resulting in non-standard systems such as intuitionistic logic, modal logics, many-valued logics, probabilistic logic, and fuzzy logic. Although such systems are certainly of interest in their own right, it now seems quite clear that formal, syntactically-based systems are not sufficient for use in a dynamic, problem-solving environment. For a further discussion of these philosophical and logical foundations, see LeFaivre[13] and McCarthy and Hayes.[15]

(2) Formal tools for representation and analysis—In the past decade a new set of formal tools and concepts for analyzing the process of approximate reasoning has evolved from the theory of fuzzy sets (Zadeh,[23] Goguen[7]). Such semantically-grounded concepts as fuzzy sets, fuzzy relations, fuzzy algorithms, linguistic variables, and linguistic hedges give us a new set of tools for symbolically representing and manipulating fuzzy information. At this time most of the work in this area has been theoretical in nature—we are just beginning to investigate the effect which some of these new concepts will have on actual problem-solving systems.

(3) Problem-specific AI systems—Although it is true that the overwhelming majority of systems developed by AI researchers have dealt with simplified domains, thereby ignoring the problem of representing real-world information, there have been some attempts to begin to come to grips with this problem. For example, Munson[18] investigated the general problem of dealing with the uncertainty encountered in the real world. Several systems which utilize semantic net memory structures provide mechanisms for handling various kinds of fuzziness (e.g., Quillian[19] and Becker[1]). Notable among these is the SCHOLAR system developed by Carbonell, into which a variety of different kinds of fuzzy reasoning capabilities have been placed (see Carbonell and Collins[5] and Collins, et al.[6]). The MYCIN system of Shortliffe[20] is also notable for its use of a model of inexact reasoning in medicine.

(4) Problem-independent AI systems—This final classification involves programming systems which are not oriented towards any particular problem domain, but rather provide general facilities for representing fuzzy knowledge. Lee[11] discusses the problem of creating a resolution theorem prover based on fuzzy logic (such a system could, in theory, be used as a general problem-solver—see Green[8]). To my knowledge, the described system was never implemented. Michalski[16] discusses the implementation of a "variable-valued logic" system which provides mechanisms for expressing and solving problems in discrete mathematics which are intrinsically nonlinear. In 1972 Rob Kling and the author began to discuss some of the modifications which might be made to a contemporary "AI language" (MICRO-PLANNER[21]) to allow it to effectively utilize fuzzy information. The resulting system (FUZZY-PLANNER) was subsequently developed on paper by Kling,[9] although it was never implemented. FUZZY-PLANNER, however, stands as the spiritual predecessor to the system described in this paper.

A BRIEF DESCRIPTION OF FUZZY

One of the more important trends in artificial intelligence in recent years has been the development of powerful new programming languages for use in AI research (see Bobrow and Raphael[4]). The research reported in this paper had as one of its goals the development of a new language which synthesized many of the good ideas of previous AI languages while providing facilities for the efficient storage, retrieval, and manipulation of fuzzy knowledge. The resulting language (called FUZZY) is a complex system which would be impossible to describe fully in a paper of this nature. I shall thus present here a brief overview of the language, and then concentrate in subsequent sections on procedural representation. Note that an early version of the system was described in Reference 12, and a complete description of the language may be found in Reference 13. A discussion of how FUZZY can be utilized to represent a variety of different forms of fuzzy knowledge, including explicitly and implicitly defined fuzzy sets, fuzzy relations, fuzzy algorithms, and linguistic hedges, appears in Reference 14. FUZZY is a LISP-based system, and has been implemented in both UNIVAC 1110 LISP and UCI LISP for the PDP-10.

Fuzzy expressions

FUZZY is in essence a many-valued programming language. By this I mean that, unlike traditional languages in which expressions evaluate to a single value, FUZZY expressions may return both a *value* and a "fuzzy truth-value," or *Z-value.** The Z-value range is specified by the LISP variables ZLOW and ZHIGH, and may be changed by the user as desired (the initial values are 0 and 1). Primitives are available for retrieving the value (VAL) or Z-value (ZVAL) portions of an expression, and for computing logical combinations of fuzzy expressions (ZAND, ZOR, ZNOT). In all cases a default Z-value of ZHIGH is provided by the system if no Z-value is present explicitly.

In addition to being able to return values and Z-values, FUZZY expressions may either *succeed* or *fail* in a manner similar to other AI languages (see Reference 4). Failure is caused by simply returning a value of FAIL, with any other value constituting success. Several variations of a standard IF-THEN-ELSE mechanism are available for controlling success and failure of individual expressions, and a user-controlled backtrack mechanism is available via the SAVE, RESTORE, and FOR statements.

The associative net

In addition to standard LISP data structures, FUZZY allows fuzzy knowledge to be explicitly represented by maintaining an associative network of *assertions* with optional Z-value modifiers. Standard ADD and REMOVE primitives are available for maintaining this net, while the FETCH primitive may be used to access the net via a powerful structural pattern-matcher. The net is automatically ordered by Z-values, and options are available in FETCH to place constraints on the Z-value range to be accessed and to specify whether the assertion with the highest or lowest Z-value in the range is to be retrieved. One variation of the FOR statement allows the user to iterate through all assertions in the net which match a certain pattern and have a Z-value in the desired range.

Fuzzy procedures

It is often the case that knowledge is complex enough to prevent it from being stored explicitly in the form of simple assertions. FUZZY thus provides a powerful method of *implicitly* representing fuzzy knowledge via the specification of procedures (using the PROC statement). The procedural mechanisms of FUZZY will be discussed more fully in a later section of this paper.

Deductive mechanisms

Like other AI languages (see Reference 4), FUZZY allows assertions to be implicitly defined via the specification of *deduce procedures.* The primitive DEDUCE is similar to FETCH, except that instead of searching for an assertion in the associative net, it searches for a procedure which allows the desired assertion to be *computed.* A GOAL primitive is also available which first performs a FETCH, utilizing DEDUCE only if the assertion is not present explicitly in the net. The pattern-directed procedure invocation mechanism of FUZZY is similar to that of other AI languages, except that FUZZY allows Z-value modifiers to be computed and returned along with assertions. Variations of the FOR statement are available for iterating through assertions generated by deduce procedures, and a simple mechanism is available for writing *generators*—procedures which compute an assertion for use by a higher-level FOR statement but which can be restarted to compute an alternative assertion if necessary.

INFERENCE IN A FUZZY ENVIRONMENT

Classical and fuzzy detachment

In order to provide some motivation for the procedural mechanisms described in the next section, let us consider briefly the problem of deductive inference in a fuzzy environment. Like other AI languages, FUZZY provides facilities for automatically "deducing" information via the use of procedures. In nonfuzzy domains, the use of deduce procedures to represent deductive arguments is relatively straightforward. For example, the classic argument "all humans are mortal; Socrates is human; therefore Socrates is mortal" may be succinctly represented by the FUZZY statements:

* The term "Z-value" was chosen so as not to give any semantic connotations to the usage of this numeric modifier, since it can represent a conventional truth-value, a fuzzy set grade of membership, a degree of certainty or belief, a simple weight, or anything else the user wishes.

```
(ADD DEDUCE:                        ; Define deduce procedure stating
 (PROC (MORTAL ?X)                  ; that humanity is sufficient to
  (GOAL (HUMAN !X)))))*             ; prove mortality.
(ADD (HUMAN SOCRATES))              ; Assert Socrates' humanity.
(DEDUCE (MORTAL SOCRATES))          ; Request proof of his mortality.
```

This argument is an example of the *rule of detachment*, which is represented in classical logic by the theorem:

(A *and* (A *implies* B)) *implies* B

In other words, given that "A" is true and "A *implies* B" is true, we may deduce that B is true.

Consider, however, the role of detachment in a nonstandard (e.g., many-valued) logic. Here the formulas "A" and "A *implies* B" may be only partially true—how then are we to compute a truth-value for B? We need to define a "detachment operator" which lets us combine the truth-values of "A" and "A *implies* B" in some way to arrive at an estimate of the truth-value of B. The importance of the selection of this operator, which is discussed more fully in Goguen,[7] Kling,[9] and LeFaivre,[13] can be illustrated by the following example.

A paradox

Consider the following two premises:

(1) 1 is a small integer.
(2) If n is a small integer, then $n+1$ is still a small integer.

Certainly these seem to be reasonable statements. We would all agree that 1 is a small integer, and surely adding 1 to a small integer doesn't magically make it not small. But given these two innocent premises, we are able to prove by an appropriate number of applications of the rule of detachment that *any* positive integer N is small. This deduction is an example of the classic "paradox of the heap" discussed by Black[2] and Goguen,[7] and illustrates one of the major shortcomings of classical logic. The problem, of course, is that the set of "small integers" is not sharply defined—we have attempted to impose the laws of classical logic on a nonclassical (i.e., fuzzy) problem.

Paradox resolved

We can make use of fuzzy detachment to resolve the above "paradox" by noting that the result of adding 1 to a small integer, while still small, is not *quite* as small as it was before. In other words, premise (2) above is not quite true—let us say that it has a truth-value of 0.99. If we assume that the truth-value of (1) is 1.0, and we use multiplication for our detachment operator, it is trivial to show that the truth-value of the statement "N is small" is 0.99^{N-1}, which is an intuitively satisfying result. Hence, the "paradox" of classical logic becomes a valid deduction when formulated in many-valued logic. Let us now consider how such a problem might be represented in FUZZY.

PROCEDURAL REPRESENTATION IN FUZZY

We saw in the preceding section an example of a problem in which a multiplicative combination of truth-values led to a satisfactory representation of a simple fuzzy deduction. It is natural to ask whether we wish to *always* combine truth-values multiplicatively. The answer seems to be no. For example, the theory of fuzzy sets makes use of the minimum and maximum for various numeric computations, and one can envision situations where the sum or average might be desired. The point is that a fuzzy problem-solving system must not force decisions of this type upon the user, but should make it easy for him to experiment with a variety of forms of control.

Global control

The issue being discussed here might be characterized as the amount of *global control* a procedure exercises over its local computations. For example, most conventional programming languages (FORTRAN, LISP) exercise no global control—all decisions (e.g., when to exit from a function) are made at the local level by statements within the procedure. On the other hand, MICRO-PLANNER[21] monitors the execution of its procedures (theorems), looking for statements which fail (return NIL) and taking some global action (backtracking) when necessary. We are faced with the question of what form of global control (if any) should be built into a FUZZY procedure. Consider a typical example: we want a procedure to succeed only if each of its local expressions succeeds with a Z-value above some threshold value. Now, this could of course be done using only local control—by inserting tests of the Z-value of each expression into the procedure—but this would clutter up the routine with rather uninteresting and repetitive local computations, obscuring its overall structure. In addition, each procedure requiring this particular global control regime would have to have the necessary local computations built in—a clearly inelegant solution. We would like to be able to simply specify the threshold value and have the

* (MORTAL ?X) acts as the *pattern* of the given procedure, which will be matched against the requested assertion in the DEDUCE statement; the sub-pattern ?X will match anything (in this case SOCRATES), binding the item matched to the variable X; !X is then used to retrieve the matched value.

system perform the necessary manipulations:

```
(PROC THRESH: ⟨n⟩ ⟨pattern⟩ ⟨e1⟩ ⟨e2⟩ . . .)
```

A fixed procedure mechanism of this form would be the FUZZY equivalent of the simple control present in MICRO-PLANNER theorems. However, as mentioned earlier, there may be other forms of global control which might be desirable—for example: ignore all failures; succeed only if at least ⟨n⟩ of the statements succeed; succeed only if some *function* of the individual Z-values exceeds a threshold (e.g., a threshold operator); return as a Z-value the minimum (maximum, sum, product) of the individual Z-values encountered; etc. FUZZY procedures allow all of these forms of global control (and more) via the specification of *procedure demons.*

Procedure demons

A procedure demon is a LISP function which is associated with a procedure, and which is given control after each expression of the procedure is evaluated. The demon is passed the result of the evaluation, the threshold value associated with the procedure, and an "accumulated Z-value" which may be dynamically computed by the demon. When the procedure is exited the demon is called one last time with a value of DONE in order to make any necessary final computations (e.g., computing an average). For example, consider the case of succeeding only if all of the expressions in a procedure succeed with a Z-value above some threshold, keeping track of the minimum Z-value encountered. A procedure demon which exercises this form of global control might be defined as follows (this is in fact FUZZY's standard default demon):

```
(DE *DEMON (V TH AC)
           (COND [(EQ V FAIL) (FAIL)]
                 [(EQ V DONE) AC]
                 [(LT (ZVAL V) TH)
                   (FAIL)]
                 [T (MIN (ZVAL V) AC)]))
```

The value returned by a procedure demon is saved by the system and becomes the new accumulated Z-value upon the next call to the demon. Thus in addition to checking for statements which fail or fall below the threshold (and causing the procedure to fail if one is found), *DEMON keeps track of the lowest Z-value encountered, as desired. The final accumulated Z-value (i.e., the value returned by the demon after it is passed DONE) is returned as the Z-value portion of the procedure result. A procedure which uses *DEMON might be defined as follows:

```
(PROC DEMON: *DEMON THRESH: 0.5
        ACCUM: 1.0 . . .)
```

This indicates that the demon for this procedure is *DEMON, the threshold value is 0.5, and the initial accumulated Z-value is 1.0 (i.e., this is the value used the first time the demon is called after entering the procedure). The DEMON:, THRESH:, and ACCUM: fields are all optional, with default values of *DEMON, ZLOW, and ZHIGH, respectively, assumed.

If a procedure demon of NIL is specified, no demon calls are made (i.e., the procedure is evaluated in a manner similar to a LISP PROG, with only local control allowed). Other procedure demons may be written by the user to specify unique forms of global control (see Reference 13). The motivation behind the procedure demon mechanism is that once a library of frequently-used procedure demons is built up, the user may easily specify a variety of global control regimes. He is thus relieved of much of the "dirty work" of directly manipulating Z-values in the programs he writes. Indeed, in many cases the user need not even be aware that he is working with fuzzy information.

Paradox revisited

Recall the "paradox" of the small integers discussed earlier. Let us briefly examine the issue of fuzzy detachment in FUZZY by showing how the procedural mechanism just described can be used to represent this problem. We first define a procedure demon DETACH which represents the (multiplicative) detachment operator:

```
(DE DETACH (V TH AC)
           (COND [(EQ V FAIL) (FAIL)]
                 [(EQ V DONE) AC]
                 [T (TIMES TH (ZVAL
                   V))]))
```

We then add the two premises to the data base. As usual, the implication "if n is small then $n+1$ is small" is expressed as a deduce procedure which utilizes backward chaining:

```
(ADD (SMALL 1))
(ADD DEDUCE:
(PROC DEMON: DETACH THRESH: 0.99
      (SMALL ?N)
      (GOAL (SMALL &(SUB1 !N))) ))
```

(The prefix "&" acts as a LISP evaluation operator). A request like:

```
(DEDUCE (SMALL 20))
```

would then cause the proper Z-value (0.99^{19}) to be computed.

Note that all of the Z-value manipulations are carried out by the procedure demon. Except for the specification of a non-standard demon and a threshold value, which in this case is analogous to the truth-value of the deduce procedure, the representation of a fuzzy deductive argument is identical to the non-fuzzy example shown previously.

CONCLUSION

One of the major goals of artificial intelligence research is the search for powerful and efficient representations of knowledge. Certain types of knowledge are adequately represented in traditional forms, e.g., as strings, arrays, or lists. In other situations net structures which are accessed associatively seem to provide more power and flexibility. In recent years it has become apparent that much knowledge is of sufficient complexity to be best represented procedurally. Recent advances in language design have provided powerful new tools which allow knowledge represented in a variety of declarative and procedural forms to be integrated in a natural manner. The research reported here was concerned with extending these representational tools in yet another dimension—toward the efficient representation and utilization of knowledge which is imprecise and uncertain in nature.

It is clear that we are a long way from being able to represent and utilize the full range of non-symbolic knowledge available to the human problem-solver in his natural reasoning processes. In the past year, however, a new representational synthesis has begun to emerge among artificial intelligence researchers (e.g., see Minsky,[17] Kuipers,[10] Winograd,[22]), although its exact nature is not yet clear. It remains to be seen what direction the search for fuzzy representations will take in light of this new synthesis.

REFERENCES

1. Becker, J., *An Information-Processing Model of Intermediate-Level Cognition,* Technical Report No. 2335, Bolt Beranek and Newman, Inc., 1970.
2. Black, M., "Reasoning with Loose Concepts," *Dialogue, 2,* pp. 1-12, 1963.
3. Bobrow, D. and A. Collins (eds.), *Representation and Understanding: Studies in Cognitive Science,* New York: Academic Press, 1975.
4. Bobrow, D. and B. Raphael, "New Programming Languages for Artificial Intelligence Research," *Computing Surveys, 6,* pp. 153-174, 1974.
5. Carbonell, J. and A. Collins, "Natural Semantics in Artificial Intelligence," *Proc. Third Int. Joint Conf. on Artificial Intelligence,* Stanford, California, 1973.
6. Collins, A., E. Warnock, N. Aiello and M. Miller, "Reasoning from Incomplete Knowledge," in Bobrow and Collins.[3]
7. Goguen, J., "The Logic of Inexact Concepts," *Synthese, 19,* pp. 325-373, 1969.
8. Green, C., *The Application of Theorem Proving to Question-Answering Systems,* Memo AI-96, Stanford University AI Lab, 1969.
9. Kling, R., "FUZZY-PLANNER: Reasoning with Inexact Concepts in a Procedural Problem-Solving Language," *J. of Cybernetics, 4,* pp. 105-122, 1974.
10. Kuipers, B., "Representing Knowledge for Recognition," in Bobrow and Collins.[3]
11. Lee, R., "Fuzzy Logic and the Resolution Principle," *J. Assoc. Comput. Mach., 19,* pp. 109-119, 1972.
12. LeFaivre, R., *FUZZY: A Programming Language for Fuzzy Problem-Solving,* Technical Report No. 202, Computer Sciences Dept., University of Wisconsin, 1974.
13. LeFaivre, R., *Fuzzy Problem-Solving,* Ph.D. Dissertation, Computer Sciences Dept., University of Wisconsin. Available as Technical Report No. 37, Madison Academic Computing Center, University of Wisconsin, 1974.
14. LeFaivre, R., "The Representation of Fuzzy Knowledge," *J. of Cybernetics, 4,* pp. 57-66, 1974.
15. McCarthy, J. and P. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in B. Meltzer and D. Michie (eds.): *Machine Intelligence 4,* Edinburgh: Edinburgh University, 1969.
16. Michalski, R., "AQVAL/1—Computer Implementation of a Variable-Valued Logic System VL/1 and Examples of Its Application to Pattern Recognition," *Proc. First Int. Joint Conf. on Pattern Recognition,* Washington, D. C., 1973.
17. Minsky, M., "A Framework for Representing Knowledge," in P. Winston (ed.): *The Psychology of Computer Vision,* New York: McGraw-Hill, 1975.
18. Munson, J., "Robot Planning, Execution, and Monitoring in an Uncertain Environment," *Proc. Second Int. Joint Conf. on Artificial Intelligence,* London, 1971.
19. Quillian, M. R., "Semantic Memory," in M. Minsky (ed.): *Semantic Information Processing,* Cambridge, MIT, 1968.
20. Shortliffe, E., *MYCIN: A Rule-Based Computer Program for Advising Physicians Regarding Antimicrobial Therapy Selection,* Memo AIM-251, Stanford University AI Lab, 1974.
21. Sussman, G., T. Winograd and E. Charniak, *Micro-Planner Reference Manual,* Memo No. 203A, MIT AI Lab, 1971.
22. Winograd, T., "Frame Representations and the Declarative-Procedural Controversy," in Bobrow and Collins.[3]
23. Zadeh, L., "Fuzzy Sets," *Information and Control, 8,* pp. 338-353, 1965.
24. Zadeh, L., "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes," *IEEE Trans. on Systems, Man, and Cybernetics, SMC-3,* pp. 28-44, 1973.
25. Zadeh, L., *The Concept of a Linguistic Variable and Its Application to Approximate Reasoning,* Memo No. ERL-M411, Electronics Research Lab, University of California, Berkeley, 1973.

Subjective Bayesian methods for rule-based inference systems*

by RICHARD O. DUDA, PETER E. HART and NILS J. NILSSON
Stanford Research Institute
Menlo Park, California

ABSTRACT

The general problem of drawing inferences from uncertain or incomplete evidence has invited a variety of technical approaches, some mathematically rigorous and some largely informal and intuitive. Most current inference systems in artificial intelligence have emphasized intuitive methods, because the absence of adequate statistical samples forces a reliance on the subjective judgment of human experts. We describe in this paper a subjective Bayesian inference method that realizes some of the advantages of both formal and informal approaches. Of particular interest are the modifications needed to deal with the inconsistencies usually found in collections of subjective statements.

INTRODUCTION

One of the characteristics of human reasoning is the ability to form useful judgments from uncertain and incomplete evidence. This ability is not only needed for everyday activities, which people would normally never formalize, but also for tasks such as medical diagnosis or securities analysis, which have been subjected to formal treatment.

Because the general need to form judgments from incomplete data is so widespread, many techniques have been developed to aid or supplant people in this task. Probability theory and statistics provide a powerful framework for dealing with many inference problems.[1,2] In standard approaches, the link between alternative hypotheses and relevant evidence is represented by conditional or joint probabilities that are estimated from statistical samples. If the number of alternative hypotheses and the amount of relevant evidence are not too great, and if the available sample is sufficiently large, then probability and statistics furnish the preferred analytical tools. However, when many kinds of evidence simultaneously bear on an hypothesis, traditional statistical approaches become inappropriate because estimation problems become unmanageable.

Recent work in artificial intelligence has suggested other approaches to the problem of resolving hypotheses on the basis of a mass of uncertain evidence. Among the most attractive are *rule-based systems,* which use a large body of *inference rules,* supplied by experts, to provide the knowledge needed to distinguish among competing hypotheses.[3-6] Each inference rule defines the role of a particular set of evidence in resolving a particular hypothesis. Typically, an ad hoc scoring function is used to combine the effects of collections of uncertain evidence acting through several inference rules on the same hypothesis. Thus, rule-based systems attempt to substitute judgments distilled from long experience for joint probabilities estimated from prohibitively large samples.

Our purpose in this paper is to describe a subjective Bayesian technique that can be used in place of ad hoc scoring functions in rule-based inference systems. Our intent is to retain insofar as possible the well-understood methods of probability theory, introducing only those modifications needed because we are dealing with networks of subjective inference rules. The scope of the paper is limited; we shall not discuss here the more general issues of representation and control that must be faced when designing a complete rule-based inference system.

FUNDAMENTALS

In a rule-based inference system, the rules are typically of the form

$$\textit{If } E_1 \textit{ and } E_2 \textit{ and } \ldots \textit{ and } E_n \quad \textit{then } H$$

where E_i $(i=1 \ldots n)$ is the i^th^ piece of evidence and H is an hypothesis suggested by the evidence. Each inference rule has a certain *strength* measured by parameters that will be defined later. For now it suffices to say that the greater the strength, the greater is the power of the evidence to confirm the hypothesis. In most applications, the rules and their strengths are provided by carefully interviewing experts.

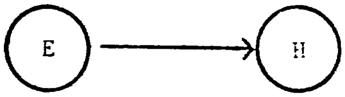
The individual pieces of evidence (the E_i) and the hypothesis (H) of a rule are propositional statements. Instead of being either absolutely true or false, the truth values of these propositional statements may be

* The work reported herein was supported by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC04-75-C-0005.

uncertain. In this paper we shall represent these uncertainties by probabilities, so that associated with each propositional statement is a corresponding probability value.

To simplify matters, we shall assume (without loss of generality) that each rule has only a single propositional statement as evidence on its left-hand side. To reduce a conjunction to a single statement, we need a method for computing the joint probability, $P(E_1, \ldots, E_n)$ from the individual probabilities $P(E_i)$. Two simple alternatives are to assume independence of the E_i or to use the fuzzy set computation $P(E_1, \ldots, E_n) = \min P(E_i)$. More generally, the left-hand side of a rule could contain an arbitrary logical expression, E. The results of this paper do not depend on how the probability of E is computed.

We represent a rule of the form "*if* E *then* H" graphically by the following structure:



Here a propositional statement is being represented as a node, and an inference rule is being represented as an arc. A collection of rules about some specific subject area invariably uses the same pieces of evidence to imply several different hypotheses. It also frequently happens that several alternative pieces of evidence imply the same hypothesis. Furthermore, there are often chains of evidences and hypotheses. For these reasons it is natural to represent a collection of rules as a graph structure or *inference net.*

An example of an inference net is shown in Figure 1.

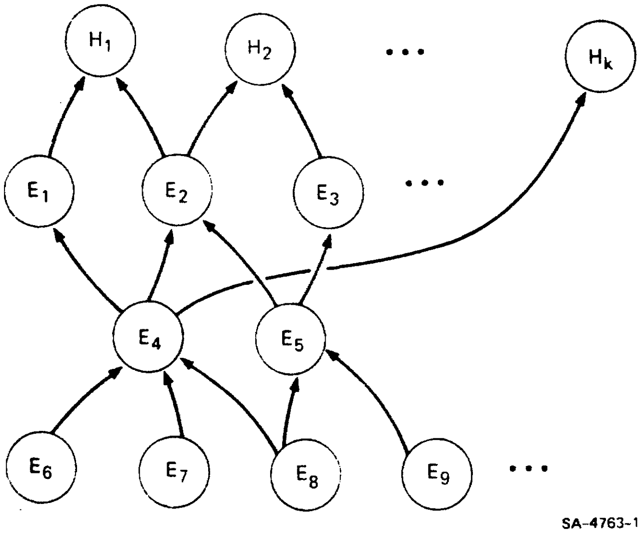


Figure 1—A simple inference net

The H_i at the top of the net are alternative hypotheses to be resolved. Each arc entering a node represents an inference rule and has associated with it a strength. Notice that a typical intermediate node like E_5 can play two roles: it provides supporting evidence for the nodes above it (E_2 and E_3), and it acts as an hypothesis to be resolved by evidence below it (E_8 and E_9).

The main problem to be considered in this paper concerns the propagation of probabilities through the net. Suppose for example, that a user of the net provides evidence by deciding that the probability of a node, say E_6, should be changed from its prior value to some new value. Obviously this should require updating of the probability of E_4 and, in turn, E_1, E_2, and H_k, and so on. Any mechanism used for propagating probabilities must be able to cope with a number of problems. The rules have uncertainty associated with them, and the evidence provided by a user may be uncertain. These two different kinds of uncertainty must somehow be combined. Multiple evidence typically bears on a single hypothesis, so that some form of independence must usually be assumed. Finally, the rules are provided subjectively by experts, so certain kinds of inconsistencies arise that can seriously jeopardize success. In the following sections we suggest a Bayesian updating scheme that addresses these concerns.

SUBJECTIVE BAYESIAN UPDATING

Suppose we are given a rule *if* E, *then* H. Let us begin with the simplified problem of updating the probability of H given its prior value and given that E is observed to be true. By Bayes rule, we have

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}. \tag{1}$$

For our purposes, a more convenient form of Bayes rule is arrived at by writing the complementary form for the negation of H

$$P(\bar{H}|E) = \frac{P(E|\bar{H})P(\bar{H})}{P(E)}, \tag{2}$$

and dividing Eq. (1) by Eq. (2) to obtain

$$\frac{P(H|E)}{P(\bar{H}|E)} = \frac{P(E|H)}{P(E|\bar{H})} \frac{P(H)}{P(\bar{H})}. \tag{3}$$

Each of the three terms in this equation has a traditional interpretation. We define the *prior odds* on H to be

$$O(H) = \frac{P(H)}{P(\bar{H})} = \frac{P(H)}{1-P(H)} \tag{4}$$

and the *posterior odds* to be

$$O(H|E) = \frac{P(H|E)}{P(\bar{H}|E)} = \frac{P(H|E)}{1-P(H|E)}. \tag{5}$$

Now the *likelihood ratio* is defined by

$$\lambda = \frac{P(E|H)}{P(E|\bar{H})}, \tag{6}$$

so Eq. 3 becomes the *odds-likelihood* formulation of Bayes rule:

$$O(H|E)=\lambda O(H). \qquad (7)$$

This equation tells us how to update the odds on H given the observation of E. For rule-based inference systems, we assume that a human expert has given the rule and has provided the likelihood ratio λ to indicate the "strength" of the rule. A high value of $\lambda (\lambda \gg 1)$ represents, roughly speaking, the fact that E is sufficient for H, since the observation that E is true will transform indifferent prior odds on H into heavy posterior odds in favor of H. Notice, incidentally, that the underlying probabilities can be recovered from their odds by the simple formula

$$P=\frac{O}{O+1}, \qquad (8)$$

so that the odds and the probabilities give exactly the same information.

Suppose now that we wish to update the odds on H given that E is observed to be false. In a strictly analogous fashion, we write.

$$O(H|\bar{E})=\bar{\lambda}O(H), \qquad (9)$$

where we define λ by

$$\lambda=\frac{P(\bar{E}|H)}{P(\bar{E}|H)}=\frac{1-P(E|H)}{1-P(E|\bar{H})}. \qquad (10)$$

Notice that $\bar{\lambda}$ must also be provided by the human expert; it cannot be derived from λ. A low value of $\bar{\lambda}$, $(0 \leq \bar{\lambda} \ll 1)$ represents, roughly speaking, the fact that E is necessary for H, since the observation that E is false will by Eq. 9 transform indifferent prior odds on H into odds heavily against H. Curiously, although λ and $\bar{\lambda}$ must be separately provided by the expert, they are not completely independent of each other. In particular, Eqs. (6) and (10) yield

$$\bar{\lambda}=\frac{1-\lambda P(E|\bar{H})}{1-P(E|\bar{H})}, \qquad (11)$$

so that, if we exclude the extreme cases of $P(E|\bar{H})$ being either 0 or 1, we see that $\lambda>1$ implies $\bar{\lambda}<1$, and $\lambda<1$ implies $\bar{\lambda}>1$. Further, we have $\lambda=1$ if and only if $\bar{\lambda}=1$. This means that if the expert gives a rule such that the presence of E enhances the odds on H (i.e., $\lambda>1$), he should also tell us that the absence of E depresses the odds on H (i.e., $\bar{\lambda}<1$). To some extent, this mathematical requirement does violence to intuition. People who work with rule-based inference systems are commonly told by experts that "The presence of E enhances the odds on H, but the absence of E has no significance." In other words, the expert says that $\lambda>1$, but $\bar{\lambda}=1$. Subsequently, we shall suggest some modifications that address this and other problems of inconsistency.

We note in passing that knowledge of both λ and $\bar{\lambda}$ is equivalent to knowledge of both $P(E|H)$ and $P(E|\bar{H})$. Indeed, it follows at once from Eqs. (6) and (10) that

$$P(E|H)=\lambda\frac{1-\bar{\lambda}}{\lambda-\bar{\lambda}} \qquad (12)$$

and

$$P(E|\bar{H})=\frac{1-\bar{\lambda}}{\lambda-\bar{\lambda}} \qquad (13)$$

Thus, whether the expert should be asked to provide λ and λ, $P(E|H)$ and $P(E|H)$, or, indeed, some other equivalent information is a psychological rather than a mathematical question.[7]

UNCERTAIN EVIDENCE AND THE PROBLEM OF PRIOR PROBABILITIES

Having seen how to update the probability of an hypothesis when the evidence is known to be either certainly true or certainly false, let us consider now how updating should proceed when the user of the system is uncertain. We begin by assuming that when a user says "I am 70 percent certain that E is true," he means that $P(E|\text{ relevant observations})=.7$. We designate by E′ the relevant observations that he makes, and simply write $P(E|E')$ for the user's response.

We now need to obtain an expression for $P(H|E')$. Formally,

$$\begin{aligned} P(H|E') &= P(H,E|E')+P(H,\bar{E}|E') \\ &= P(H|E,E')P(E|E') \\ &\quad +P(H|\bar{E},E')P(\bar{E}|E'). \end{aligned} \qquad (14)$$

We make the reasonable assumption that if we *know* E to be true (or false), then the observations E′ relevant to E provide no further information about H. With this assumption, Eq. (14) becomes

$$P(H|E')=P(H|E)P(E|E')+P(H|\bar{E})P(\bar{E}|E'). \qquad (15)$$

Here $P(H|E)$ and $P(H|\bar{E})$ are obtained directly from Bayes rule, i.e., from Eq. (7) and Eq. (9), respectively.

If the user is certain that E is true, then $P(H|E')=P(H|E)$. If the user is certain that E is false, then $P(H|E')=P(H|\bar{E})$. In general, Eq. (15) gives $P(H|E')$ as a linear interpolation between these two extreme cases. In particular, note that if $P(E|E')=P(E)$ then $P(H|E')=P(H)$. This has the simple interpretation that if the evidence E′ is no better than a priori knowledge, then application of the rule leaves the probability of H unchanged.

In a pure Bayesian formulation, Eq. (15) is the solution to the updating question. In practice, however, there are significant difficulties in using this formulation in an inference net. These difficulties stem from a combination of the classical Bayesian dilemma over prior probabilities and the use of subjective probabilities.

To appreciate the difficulty, consider again a typical pair of nodes E and H embedded in an inference net. It is apparent from Eqs. (7) and (9) that the updating procedure depends on the availability of the prior odds

O(H). Thus, although we have not emphasized the point until now, we see that the expert must be depended upon to provide the prior odds as well as λ and $\bar{\lambda}$ when the inference rule is given. On the other hand, recall our earlier observation that E also acts as an hypothesis to be resolved by the nodes below it in the net. Thus, the expert must also provide prior odds on E. If all of these quantities were specified consistently, then the situation would be as represented in Figure 2. The straight line plotted is simply Eq. (15), and shows the interpolation noted above. In particular, note that if the user asserts that $P(E|E')=P(E)$, then the updated probability is $P(H|E')=P(H)$. In other words, if the user provides no new evidence, then the probability of H remains unchanged.

In the practical case, unfortunately, the subjectively obtained prior probabilities are virtually certain to be inconsistent, and the situation becomes as shown in Figure 3. Note that P(E), the prior probability provided by the expert, is different from $P_c(E)$, the probability consistent with P(H). Here, if the user provides no new evidence—i.e., if $P(E|E')=P(E)$—then the formal Bayesian updating scheme will substantially change the probability of H from its prior value P(H). Furthermore, for the case shown in Figure 3, if the user asserts that E is true with a probability $P(E|E')$ lying in the interval between P(E) and $P_c(E)$, then the updated probability $P(H|E')$ will be less than P(H). Thus, we have here an example of a rule intended to increase the probability of H if E is found to be true, but which turns out to have the opposite effect. This type of error can be compounded as probabilities are propagated through the net.

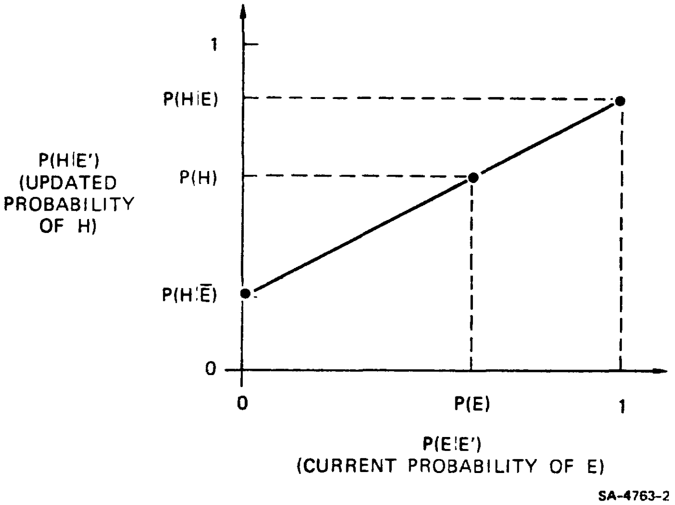


Figure 2—Idealized updating of $P(H|E')$

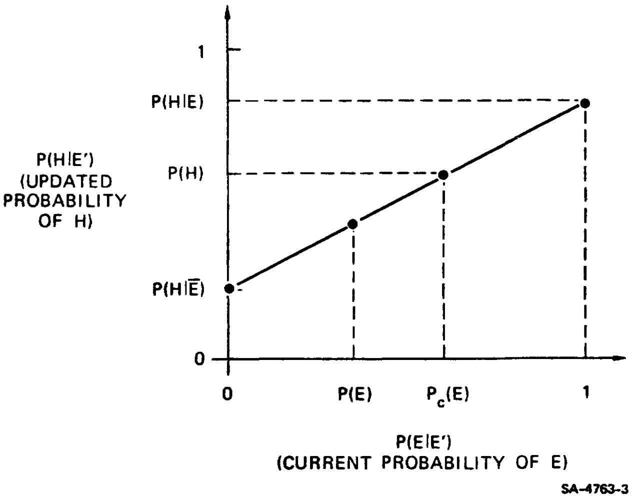


Figure 3—Inconsistent priors

Several measures can be taken to correct the unfortunate effects of priors that are inconsistent with inference rules. Since the problem can be thought of as one of overspecification, one approach would be to relax the specification of whatever quantities are subjectively least certain. For example, if the subjective specification of P(E) were least certain (in the expert's opinion), then we might set $P(E)=P_c(E)$. This approach leads to difficulties because the pair of nodes E and H under consideration are embedded in a large net. For example, in Figure 1, we might be considering node E_2 as the hypothesis H, and node E_5 as the evidence E. If we were to establish a prior probability $P(E_5)$ to be consistent with $P(E_2)$, we would simultaneously make $P(E_5)$ inconsistent with the priors on E_8 and E_9, which provide supporting evidence for E_5. Prior probabilities can therefore not be forced into consistency on the basis of the local structure of the inference net; apparently, a more global process—perhaps a relaxation process—would be required.

A second alternative for achieving consistency would be to adjust the linear interpolation function shown in Figure 3. There are several possibilities, one of which is illustrated in Figure 4a. The linear function has been broken into a piecewise linear function at the coordinates of the prior probabilities, forcing consistent updating of the probability of H given E′. Two other possibilities are shown in Figures 4b and 4c. In Figure 4b we have introduced a dead zone over the interval between the specified prior probability P(E) and the consistent prior $P_c(E)$. Intuitively, the argument in support of this consistent interpolation function is that if the user cannot give a response outside this interval, then he is not sufficiently certain of his response to warrant any change in the probability of H. Figure 4c shows another possibility, motivated by the earlier observation that experts often give rules of the form "The presence of E enhances the odds on H, but the absence of E has no significance." By keeping $P(H|E')$ equal to P(H) when $P(E|E')$ is less than P(E) we are

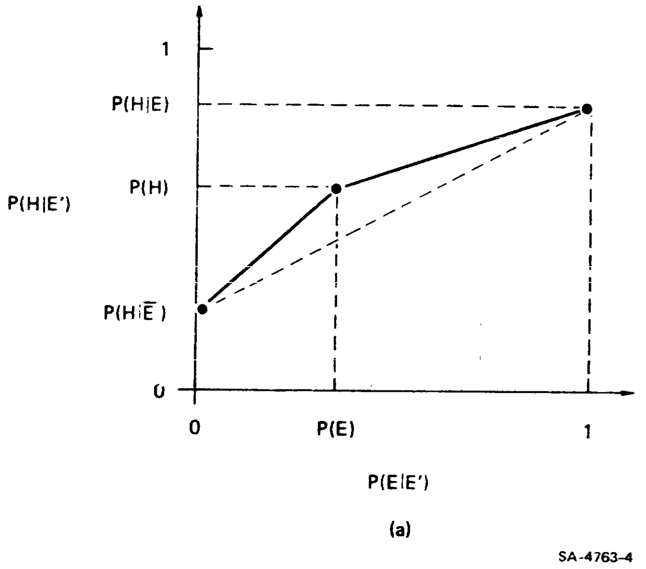


Figure 4(a)—Consistent interpolation functions

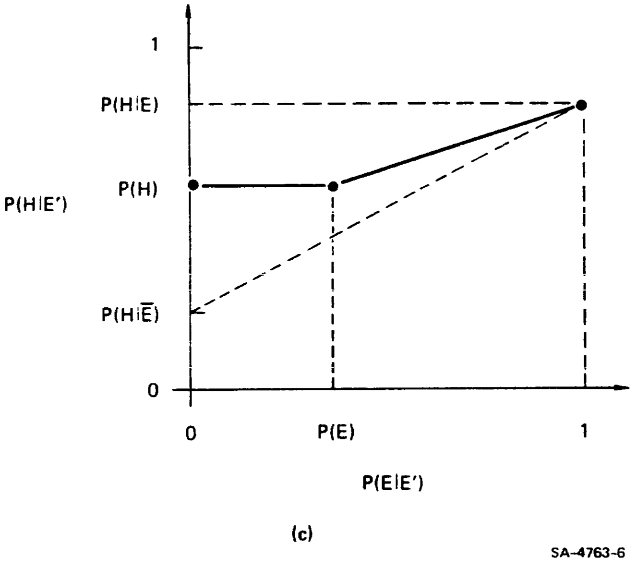


Figure 4(c)—Consistent interpolation functions (concluded)

effectively allowing the forbidden situation where $\lambda > 1$ and $\bar{\lambda} = 1$. In effect, this is equivalent to the method illustrated in Figure 4a under the assumption that $P(H|\bar{E}) = P(H)$.

It is interesting to compare these modifications with the procedure used by Shortliffe to handle uncertain evidence in the MYCIN system.[4,5] While the nonlinear equations that result from use of Shortliffe's version of confirmation theory prevent a general comparison, it is possible to express his procedure in our terms for the special case of a single rule. The result for the case in which the presence of E supports H is shown in Figure 5. Clearly, the solution is identical to that of Figure 4c except for the interval from P(E) to $P_t(E)$ within which Shortliffe's solution maintains $P(H|E')$ at the a priori value P(H).

The graphical representations in Figures 2 through 4 provide a nice vehicle for visualizing the discrepancies between formal and subjective Bayesian updating, and make it easy to invent other alternatives for reconciling inconsistencies. For completeness, the Appendix contains the easily computable algebraic representa-

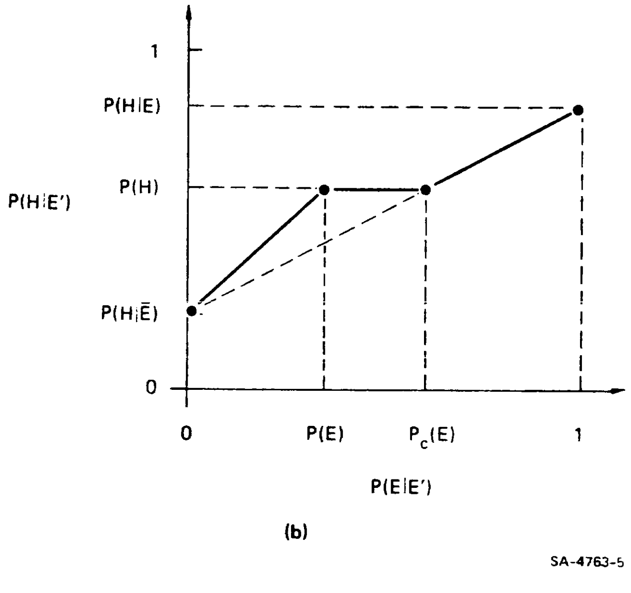


Figure 4(b)—Consistent interpolation functions (continued)

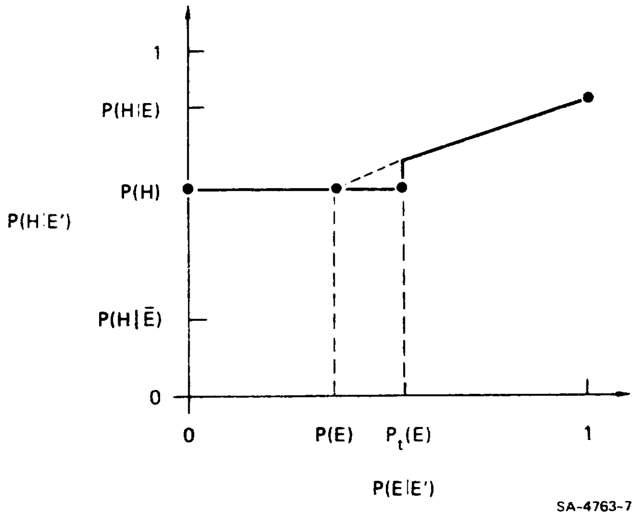


Figure 5—The interpolation function used in the mycin system $P_t(E) = P(E) + t[1 - P(E)]$. Typically, t=0.2.

tions of these functions, and also treats the complementary case in which the straight line given by Eq. (15) has a negative slope (the case in which $\lambda<\bar{\lambda}$). In a small experimental system, the function shown in Figure 4a has given satisfactory preliminary results.[5]

THE USE OF MULTIPLE EVIDENCE

We turn now to the more general updating problem in which several rules of the form $E_1 \rightarrow H, \ldots, E_n \rightarrow H$ all concern the same hypothesis H.* Since most nodes in actual inference nets have several incoming arcs, this is the case of greatest practical interest. In order to gain some insight about how multiple evidence should be used to update H when the evidence is uncertain and the priors are inconsistent, let us first consider briefly how updating would formally proceed in simpler cases.

Suppose the ith inference rule has associated with it the usual two quantities λ_i and $\bar{\lambda}_i$. For a first simple case, how should H be updated when all the E_i have been observed to be certainly true? This case is analogous to the case summarized by Eq. (7). Under the assumption that the pieces of evidence are conditionally independent (i.e., that $P(E_1, \ldots, E_n|H) = \prod_{i=1}^{n} P(E_i|H)$ and that $P(E_1, \ldots, E_n|\bar{H}) = \prod_{i=1}^{n} P(E_i|\bar{H})$), it is not difficult to reach an analogous answer. Specifically, the odds on H are updated by the expression

$$O(H|E_1, \ldots, E_n) = \left[\prod_{i=1}^{n} \lambda_i\right] O(H), \qquad (16)$$

where

$$\lambda_i = \frac{P(E_i|H)}{P(E_i|\bar{H})}. \qquad (17)$$

Similarly, if all the evidence is observed to be certainly false, we can under conditional independence assumptions again factor the joint likelihood ratio to obtain

$$O(H|\bar{E}_1, \ldots, \bar{E}_n) = \left[\prod_{i=1}^{n} \bar{\lambda}_i\right] O(H). \qquad (18)$$

Now let us consider the general case of uncertain evidence and inconsistent prior probabilities. We already know that the posterior odds $O(H|E_i')$ given a single observation E_i' can be computed using updating functions like the ones shown in Figure 4. We can therefore define, for a single inference rule, an effective likelihood ratio λ_i' by

$$\lambda_i' \stackrel{\Delta}{=} \frac{O(H|E_i')}{O(H)}. \qquad (19)$$

By making the assumption now that the E_i' are independent, we can obtain for the general case an expression similar to the simple updating formulas given by Eqs. (16) and (18):

$$O(H|E_1', \ldots, E_n') = \left[\prod_{i=1}^{n} \lambda_i'\right] O(H). \qquad (20)$$

To use this expression in an inference net system, we simply store with each node its prior odds (or probability), and store with each incoming arc an effective likelihood ratio λ_i'. Whenever a piece of evidence provided by the user causes $P(E_i|E_i')$ to be updated, a new effective likelihood ratio is computed and the posterior odds in favor of H is computed using Eq. (20). This procedure has the following consequences:

(1) If no evidence is obtained for a rule, then it will retain an initial effective likelihood ratio of unity, since prior and "posterior" odds are the same.

(2) The order in which evidence is obtained and rules are applied does not affect the final posterior probabilities.

(3) The same rule can be used repeatedly, with the same or different values for the probability of the evidence. In particular, if a user changes his mind and modifies an earlier assertion, the new assertion will correctly "undo" any effects of earlier statements.

CONCLUSIONS

The probability updating procedure presented here has several points to recommend it. It accepts subjective information that can readily be obtained from experts. The two conditional probabilities, $P(E|H)$ and $P(E|\bar{H})$, that determine the strength of an inference rule typically are intuitively meaningful measures, and the procedure is tolerant of the inevitable inconsistencies in subjective expert information. The basis in probability theory of our procedure provides a useful theoretical foundation for calculating the effects of uncertain evidence. One value of theory is that it makes us explicitly aware of certain underlying assumptions about such matters as conditional independence, prior probabilities, and inconsistent information. Finally, our procedure is straightforward computationally and can be readily implemented in inference net systems.

There are, however, some questions that remain to be dealt with. If the network contains multiple paths linking a given piece of evidence to the same hypothesis, the independence assumption is obviously violated. It is important to settle on a reasonable (if ad hoc) modification of our basic procedure that behaves appropriately in such situations. (A more extreme complication would involve being able to avoid the circular reasoning implied by inference nets with loops.)

* This should not be confused with the conjunctive premise mentioned earlier.

There are sometimes cases where some of the nodes in an inference net are related by a constraint not expressed in any given rule. For example, a subset of hypotheses may be mutually exclusive and exhaustive, in which case their probabilities must always sum to one, regardless of their individual values. Such a constraint may be inconsistent with the associated rule strengths given us by the experts. Perhaps a simple expedient, such as renormalization of probability values, can be justified in this case.

We have not addressed here at all issues of inference net control strategy: for example, which hypotheses should be pursued and which evidence should be sought at any step. The answers to these sorts of questions may be heavily dependent on the particular application. Another global question concerns rules containing logical statements that may include quantifiers and variables. But in whatever way these questions are answered, the basic updating procedure presented here would appear to be a useful component of rule-based inference systems.

ACKNOWLEDGMENTS

We have benefited from the comments of many of our colleagues, but would like particularly to acknowledge the contributions of Georgia Sutherland at SRI, and the stimulating and helpful discussions with E. H. Shortliffe, Bruce Buchanan, Randall Davis, and Dana Ludwig at Stanford University. This work was supported by the Advanced Projects Research Agency under Contract DAHCO4-75-C-0005.

REFERENCES

1. Hadley, G., *Introduction to Probability and Statistical Decision Theory,* Holden-Day, San Francisco, California, 1967.
2. Raiffa, H., *Decision Analysis,* Addison-Wesley, New York, New York, 1968.
3. Waterman, D. A., "Generalization Learning Techniques for Automating the Learning of Heuristics," *Artificial Intelligence,* Vol. 1, pp. 121-170, Spring 1970.
4. Shortliffe, E. H., *MYCIN: A Rule-Based Computer Program for Advising Physicians Regarding Antimicrobial Therapy Selection,* Stanford Artificial Intelligence Laboratory Memo AIM-251, Stanford University, Stanford, California, October 1974.
5. Shortliffe, E. H. and B. G. Buchanan, "A Model of Inexact Reasoning in Medicine," *Mathematical Biosciences,* Vol. 23, pp. 351-379, 1975.
6. Davis, R. and J. King, "An Overview of Production Systems," in *Machine Representations of Knowledge,* D. Reidel Publishing Co.; forthcoming.
7. Gustafson, D. H., et al., "Wisconsin Computer Aided Medical Diagnosis Project—Progress Report," in *Computer Diagnosis and Diagnostic Methods,* pp. 255-278, J. A. Jacquez, ed., Charles C. Thomas, Springfield, Illinois, 1972.
8. Sutherland, G., *Implementation of Inference Nets—II,* Technical Note 122, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, January 1976.

APPENDIX

Complete analytical expressions giving $P(H|E')$ as a piecewise linear function of $P(E|E')$ are given in this Appendix. These expressions correspond to the three graphical representations illustrated in Figure 4. The simplest expression corresponds to Figure 4a:

$$P(H|E') = \begin{cases} P(H|\bar{E}) + \dfrac{P(E|E')}{P(E)}[P(H) - P(H|\bar{E})] & O \leq P(E|E') \leq P(E) \\ \dfrac{P(H) - P(H|E)P(E)}{1-P(E)} + P(E|E')\,\dfrac{P(H|E) - P(H)}{1-P(E)} & P(E) \leq P(E|E') \leq 1 \end{cases} \tag{A1}$$

Here it is important to note that the four quantities $P(H)$, $P(E)$, $P(H|E)$, and $P(H|\bar{E})$ are assumed to be estimates obtained from experts. Were the true probabilities to be used in this formula, it would reduce at once to the linear expression given by Eq. (15). The estimates of $P(H|E)$ and $P(H|\bar{E})$ might be obtained directly from an expert, but would more often be obtained through Bayes rule [Eqs. (7) and (9), respectively]. To be explicit,

$$P(H|E) = \frac{P(E|H)P(H)}{[P(E|H) - P(E|\bar{H})]P(H) + P(E|\bar{H})} = \frac{\lambda P(H)}{(\lambda-1)P(H)+1} \tag{A2}$$

and

$$P(H|\bar{E}) = \frac{[1-P(E|H)]P(H)}{[P(E|\bar{H}) - P(E|H)]P(H) + 1 - P(E|\bar{H})} = \frac{\bar{\lambda}P(H)}{(\lambda-1)P(H)+1} \tag{A3}$$

To obtain the equations for Figure 4b, we define $P_c(E)$ by

$$P_c(E) = \frac{P(H) - P(H|\bar{E})}{P(H|E) - P(H|\bar{E})} \tag{A4}$$

In general, this quantity will differ from the $P(E)$ value supplied by the expert. For Figure 4b we must distinguish between the two cases $P(E) \leq P_c(E)$ and $P(E) > P_c(E)$. The equations are as follows:

Case 1: $P(E) \leq P_c(E)$

$$P(H|E) = \begin{cases} P(H|\bar{E}) + \dfrac{P(E|E')}{P(E)}[P(H) - P(H|\bar{E})] & 0 \leq P(E|E') \leq P(E) \\ P(H) & P(E) \leq P(E|E') \leq P_c(E) \\ P(H|\bar{E}) + P(E|E')\,[P(H|E) - P(H|\bar{E})] & P_c(E) \leq P(E|E') \leq 1 \end{cases} \tag{A5}$$

Case 2: $P(E) > P_c(E)$

$$P(H|E') = \begin{cases} P(H|\bar{E}) + P(E|E')\,[P(H|E) - P(H|\bar{E})] & 0 \leq P(E|E') \leq P_c(E) \\ P(H) & P_c(E) \leq P(E|E') \leq P(E) \\ \dfrac{P(H) - P(H|E)P(E)}{1-P(E)} + P(E|E')\dfrac{P(H|E) - P(H)}{1-P(E)} & P(E) \leq P(E|E') \leq 1 \end{cases} \tag{A6}$$

Finally, there are also two cases to be distinguished for Figure 4c. The first case corresponds to assuming that $P(H|\bar{E}) \approx P(H)$, so that $P_c(E) \approx 0$. The second case corresponds to assuming that $P(H|E) \approx P(H)$, so that $P_c(E) \approx 1$. In effect, these cases correspond to the rules $E \xrightarrow{\lambda} H$ and $\bar{E} \xrightarrow{\bar{\lambda}} H$ taken separately. The corresponding equations are special cases of Eqs. (A5) and (A6):

Case 1: $E \xrightarrow{\lambda} H$

$$P(H|E') = \begin{cases} P(H) & 0 \leq P(E|E') \leq P(E) \\ \dfrac{P(H) - P(H|E)P(E)}{1-P(E)} + P(E|E')\,\dfrac{P(H|E) - P(H)}{1-P(E)} & P(E) \leq P(E|E') \leq 1 \end{cases} \tag{A7}$$

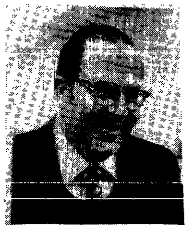
Case 2: $\bar{E} \xrightarrow{\bar{\lambda}} H$

$$P(H|E') = \begin{cases} P(H|\bar{E}) + \dfrac{P(E|E')}{P(E)}[P(H) - P(H|\bar{E})] & 0 \leq P(E|E') \leq P(E) \\ P(H) & P(E) \leq P(E|E') \leq 1 \end{cases} \tag{A8}$$

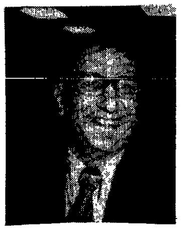
Ordinarily one would view this as a simplified approximation that is useful when one of the two likelihood ratios is dominant. However, it is interesting to observe that if both λ and $\bar{\lambda}$ are significant and if the two separate rules $E \xrightarrow{\lambda} H$ and $\bar{E} \xrightarrow{\bar{\lambda}} H$ are treated as if E and $\bar{E}$ were statistically independent, then Eqs. (A7) and (A8) yield the same result as Eq. (A1). This follows from the fact that when $P(H|E') = P(H)$ we have $O(H|E') = O(H)$, so that Eq. (19) yields $\lambda' = 1$. Thus, if $0 \leq P(E|E') \leq P(E)$ only the rule $\bar{E} \xrightarrow{\bar{\lambda}} H$ contributes to $P(H|E')$, while if $P(E) \leq P(E|E') \leq 1$ only the rule $E \xrightarrow{\lambda} H$ contributes to $P(H|E')$, the contributions being exactly those given in Eq. (A1).

1976 NATIONAL COMPUTER CONFERENCE COMMITTEES

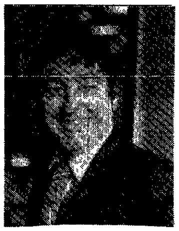
CONFERENCE
Carl Hammer
Sperry Univac
Washington, DC



PROGRAM
Stanley Winkler
IBM Corporation
Gaithersburg, MD



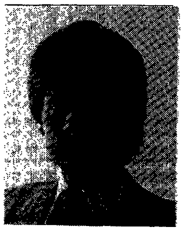
FINANCE
Norman Moraff
Bureau of the Census
Suitland, MD



SPECIAL ASSISTANT
Lee Danner
IBM Corporation
Gaithersburg, MD

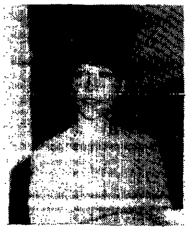
PUBLICATIONS
Joyce A. Amenta
Informatics, Inc.
Rockville, MD

PUBLIC RELATIONS
Dorothy Ray
General Research Corporation
McLean, VA



SPECIAL ACTIVITIES
Thomas A. D'Auria
Columbia University
New York, NY

CONFERENCE OPERATIONS
Ruben D. Maldonado
Auerbach Associates
Germantown, MD



STUDENT COMPUTER FAIR
Sema Marks
IBM Corporation
New York, NY

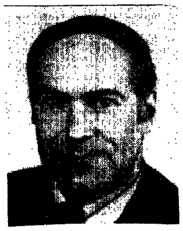
COMPUTER GRAPHICS ART EXHIBIT
Jackie Potts
Social Security Administration
Baltimore, MD

SCIENCE FILM THEATER
Adrian J. Basili
A.T.&T. Co.
New York, NY

LEGAL COUNSEL
Ronald L. Winkler
Sutherland, Asbill & Brennan
Washington, DC

BICENTENNIAL ACTIVITIES
Edward K. Zimmerman
Bicentennial Administration
Washington, DC

EXHIBITS
Thomas W. Johnston
Control Data Corporation
Minneapolis, MN



NCCC LIAISON
Harvey Garner
University of Pennsylvania
Philadelphia, PA

NCCC LIAISON
Henry S. McDonald
Bell Laboratories
Murray Hill, NJ

HISTORICAL PERSPECTIVE
William F. Luebbert
U.S. Military Academy
West Point, NY

AWARDS
Marjorie Jewell
AFIPS
Montvale, NJ

1924-1975

SECRETARY
Cecil Shelton

PROGRAM COMMITTEE

Chairman

Stanley Winkler
IBM Corporation
Gaithersburg, MD

David H. Ahl
Creative Computing
Morristown, NJ

John L. Berg
National Bureau of Standards
Washington, DC

Robert A. Beverage
IBM Corporation
Gaithersburg, MD

Robert L. Brueck
MRI Systems Corporation
Austin, TX

Dennis Branstad
National Bureau of Standards
Washington, DC

Anita J. Cochran
Bell Laboratories
Murray Hill, NJ

Ira Cotton
National Bureau of Standards
Washington, DC

Philip H. Enslow, Jr.
Georgia Institute of Technology
Atlanta, GA

Saul I. Gass
University of Maryland
College Park, MD

Robert Gildea
The Mitre Corporation
Colorado Springs, CO

Harvey J. Greenberg
Virginia Polytechnic Institute
Reston, VA

Anne M. Gulick
IBM
Gaithersburg, MD

Franklin F. Kuo
Department of Defense
Washington, DC

Herbert Maisel
Georgetown University
Washington, DC

Stuart L. Mathison
Telenet Communication Corporation
Washington, DC

David Mishelevich
The University of Texas Health Science Ctr.
Dallas, TX

Norman Moraff
Bureau of the Census
Suitland, MD

Thomas Murray
DelMonte Corporation
San Francisco, CA

George N. Nomicos
IBM Education Center-Europe
Brussells, Belgium

Paul Oliver
Department of the Navy
Washington, DC

Norman Rasmussen
Consultant
Boston, MA

Dan C. Ross
Ross Telecommunications Engineering Corp.
Washington, DC

J. E. Savage
Brown University
Providence, Rhode Island

Philip M. Walker
Telenet Communication Corporation
Washington, DC

Milton J. Waxman
Systems Group of TRW, Inc.
Redondo Beach, CA

Kurt J. Ziegler, Jr.
IBM Corporation
Gaithersburg, MD

PROGRAM ADVISORY COMMITTEE

Maybelle Cremer
Tuscon, AZ

Ruth M. Davis
National Bureau of Standards
Washington, DC

Alexander S. Douglas
The London School of Economics
and Political Science
University of London
London, England

Edward J. Grenier, Jr.
Sutherland, Asbill & Brennan
Washington, DC

Janice C. Lipsen
Counselors for Management
Washington, DC

Nathaniel Macon
The American University
Washington, DC

Henry S. McDonald
Bell Laboratories
Murray Hill, NJ

Stephen W. Miller
Stanford Research Institute
Menlo Park, CA

Jack Moshman
Moshman Associates, Inc.
Washington, DC

Bruce G. Oldfield
IBM Corporation
Paris, France

Heinz Zemanek
IBM Corporation
Vienna, Austria

FINANCE COMMITTEE

Chairman

Norman Moraff
Bureau of the Census
Suitland, MD

Joan Golden
Old Bridge, NJ

SPECIAL ASSISTANT'S COMMITTEE

Chairman

Lee Danner
IBM Corporation
Gaithersburg, MD

Robert G. Abbott
Yourdon, Inc.
New York, NY

Marshal D. Abrams
National Bureau of Standards
Gaithersburg, MD

Terrence H. Coyle
Eastman Kodak Company
Rochester, NY

Verna Danner
Computer Learning Center
Fairfax, VA

Raymond G. Fox
IBM Corporation
Manassas, VA

Delores C. Harris
IBM Corporation
Gaithersburg, MD

John H. Mitchell
Eastman Kodak Company
Rochester, NY

Paul D. Oyer
U.S. Bureau of the Census
Washington, DC

Harold J. Podell
U.S. General Accounting Office
Washington, DC

Walter E. Simonson
U.S. Bureau of Census
Washington, DC

Rosetta L. Winkler
Consultant
Bethesda, MD

Edward Yourdon
Yourdon, Inc.
New York, NY

PUBLICATIONS COMMITTEE

Chairman

Joyce A. Amenta
Informatics, Inc.
Rockville, MD

Lester Bounds
Lester Bounds, Inc.
Arlington, VA

Kenneth J. McCallister
Federal City College
Washington, DC

Paul D. Oyer
U.S. Bureau of the Census
Washington, DC

Dorothy Ray
General Research Corporation
McLean, VA

Lester Ungerleider
AAI
Arlington, VA

Virginia Walker
U.S. Bureau of the Census
Washington, DC

PUBLIC RELATIONS COMMITTEE

Chairman

Dorothy Ray
General Research Corporation
McLean, VA

Karl R. Ahren
James Talcott Inc.
New York, NY

J. D. Alexander
Fleet Material Support Office
Mechanicsburg, PA

Byron Allen
Air Force Avionics Labs
Dayton, OH

Jim Brandeberry
Wright State University
Dayton, OH

James Case
Worldwide Church of God
Pasadena, CA

Vice-Chairman
Nancy L. Ayer
ADP Systems
Falls Church, VA

Joe K. Clema
General Dynamics
Dayton, OH

Robin Connelly
Worldwide Church of God
Pasadena, CA

R. Crisafulli
Education Testing Service
Princeton, NJ

Carson Grabbe
Worldwide Church of God
Pasadena, CA

Charles E. Green
Renegotiation Board
Washington, DC

Frank Hubans, Jr.
General Dynamics
Dayton, OH

Susan H. Lewis
Signal Processing Systems, Inc.
Waltham, MA

Ruth McQueen
Amarillo College
Amarillo, TX

Annamary M. Phillips
Raytheon Company
Burlington, MA

Krishnan Ramaswamy
Pennsylvania Department of Health
Harrisburg, PA

David R. Skeen
Office of Naval Research
Arlington, VA

Thomas J. Sorger
Sorger Associates
Peabody, MA

Russell Staley
Texas A&M University
College Station, TX

Ronald Stewart
Systems Design Consultants
Des Plaines, IL

Vern Van Dyke
National Agricultural Library
Beltsville, MD

Alan Zimmermann
Computer Processing Institute
East Hartford, CT

Murray Zuckerman
Insurance Systems Consultants
San Francisco, CA

SPECIAL ACTIVITIES COMMITTEES

Chairman

Thomas A. D'Auria
Columbia University
New York, NY

David H. Brandin
Stanford Research Institute
Menlo Park, CA

Donna Denyer
New York Times
New York, NY

Mimi Garrard
Mimi Garrard Dance Company
New York, NY

Steven L. Jamison
IBM Corporation
Palo Alto, CA

Lou Katz
Columbia University
New York, NY

Hal Lamster
Telmar Communications
New York, NY

Nancy Mackta
American Express Company
New York, NY

Sema Marks
IBM Corporation
New York, NY

Douglas Rochester
Chase Manhattan Bank
New York, NY

Robert Wine
Chase Manhattan Bank
New York, NY

CONFERENCE OPERATIONS COMMITTEE

Chairman

Ruben D. Maldonado
Auerbach Associates
Germantown, MD

Claire Chase
New York City Finance Administration
New York, NY

Lori Capadanno
American Telephone & Telegraph
Basking Ridge, NJ

Pat Cunniff
Chase Manhattan Bank
New York, NY

Tom D'Auria
Columbia University
New York, NY

Ed Gittleson
New York City Human Resources Agency
New York, NY

Warren Person
Municipal Credit Union
New York, NY

STUDENT COMPUTER FAIR COMMITTEE

Chairman

Sema Marks
IBM Corporation
New York, NY

David Ahl
Creative Computing
Morristown, NJ

Joseph Gianrotti
City University of New York
New York, NY

Laurence Heimrath
City University of New York
New York, NY

Beth Norman
City University of New York
New York, NY

Howard Rubin
Hunter College
New York, NY

Robert Taylor
Teachers College
New York, NY

Robert G. Wine
The Chase Manhattan Bank
New York, NY

STUDENT COMPUTER FAIR JUDGES

Joel S. Birnbaum
IBM Research Division
Yorktown Heights, NY

Ludwig Braun
SUNY at Stony Brook
Stony Brook, NY

Bill Etra
University of Maryland
Baltimore, MD

Louise Etra
City University of New York
New York, NY

Steve Gray
Consultant
Darien, CT

Nancy Grosch
Information Systems Design
Santa Clara, CA

Walter Koetke
Lexington High School
Lexington, MA

Stephen Levine
Lawrence Livermore Lab
Livermore, CA

Daniel McCracken
Consultant
Ossining, NY

Ted Nelson
University of Illinois
Chicago, IL

Seymour Papert
Massachusetts Institute of Technology
Cambridge, MA

Frederik Pohl
Bantam Books, Inc.
New York, NY

COMPUTER GRAPHICS ART EXHIBIT COMMITTEE

Chairman

Jackie Potts
Social Security Administration
Baltimore, MD

Alyce Branum
Digital Equipment Corporation
Marlboro, MA

George Champine
Univac
Roseville, MN

William Fetter
Southern Illinois University
Carbondale, IL

Grace Hertlein
California State University
Chico, CA

Ken Knowlton
Bell Laboratories
Murray Hill, NJ

Kurt F. Lauckner
Eastern Michigan University
Ypsilanti, MI

E. T. Manning
Watson-Manning, Inc.
Stratford, CT

SCIENCE FILM THEATRE COMMITTEE

Chairman

Adrian J. Basili
American Telephone & Telegraph
New York, NY

Rhonda Beck
Western Electric
New York, NY

Robert C. Spieker
Western Electric
New York, NY

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES, INC. (AFIPS)

OFFICERS AND BOARD OF DIRECTORS

President

Anthony Ralston
State University of New York
Amherst, NY

Vice President

Albert S. Hoagland
IBM Corporation
San Jose, CA

Secretary

Sylvia Charp
The School District of Philadelphia
Philadelphia, PA

Treasurer

Walter A. Johnson
Consolidated Paper Company, Inc.
Wisconsin Rapids, WI

Executive Director

Robert W. Rector
AFIPS
Montvale, NJ

ACM Directors

Jean Sammet
IBM Corporation
Cambridge, MA

Willard J. Holden
Palo Alto, CA

Stuart Lynn
Rice University
Houston, TX

IEEE Directors

Dick Simmons
Texas A&M
College Station, TX

Stephen S. Yau
Northwestern University
Evanston, IL

Rolland B. Arndt
Sperry Univac
St. Paul, MN

Data Processing Management Association Directors

Edward J. Palmer
Boston University
Boston, MA

Herbert B. Safford
GTE Data Services, Inc.
Marina Del Rey, CA

Institute of Internal Auditors Director

William E. Perry
The Institute of Internal Auditors
Orlando, FL

Society for Computer Simulation Director

Paul W. Berthiaume
Electronic Associates, Inc.
West Long Branch, NJ

Society for Information Display Director

Carlo P. Crocetti
Rome Air Development Center
Griffis Air Force Base, NY

Association for Educational Data Systems Director

Thomas McConnell
Atlanta Public Schools
Atlanta, GA

American Institute of Certified Public Accountants Director

John Mitchell
American Institute of Certified Public Accountants
New York, NY

American Statistical Association Director

James Filliben
National Bureau of Standards
Washington, DC

Instrument Society of America Director

Theodore J. Williams
Purdue University
W. Lafayette, IN

Society for Industrial and Applied Mathematics Director

Donald L. Thomsen, Jr.
SIAM Institute for Mathematics
New Canaan, CT

Special Libraries Association Director

Herbert S. White
Graduate Library, School University
Bloomington, IN

American Institute of Aeronautics and Astronautics Director

Kenneth A. Hales
Boeing Aerospace Co.
Seattle, WA

American Society for Information Science Director

Harold Borko
UCLA School of Library Science
Los Angeles, CA

Association for Computation Linguistics Director

A. Hood Roberts
Center for Applied Linguistics
Arlington, VA

NATIONAL COMPUTER CONFERENCE BOARD MEMBERS

Chairman-IEEE-CS Representative

Merlin Smith
T.J. Watson Research Center
Yorktown Heights, NY

Vice Chairman—SCS Representative

Ralph Wheeler
LMSC
Sunnyvale, CA

Treasurer-AFIPS Representative

Walter A. Johnson
Consolidated Paper Company, Inc.
Wisconsin Rapids, WI

Secretary-AFIPS Representative

Theodore J. Williams
Purdue University
West Lafayette, IN

AFIPS Representatives

Anthony Ralston
State University of New York
Amherst, NY

Albert S. Hoagland
IBM Corporation
San Jose, CA

DPMA Representative

James F. Towsen
The Statesman Group
Harrisburg, PA

ACM Representative

Smith Dorsey
Rockwell International
Anaheim, CA

NATIONAL COMPUTER CONFERENCE COMMITTEE

Chairman

Jeffrey D. Stein
On-Line Business Systems, Inc.
San Francisco, CA

Morton M. Astrahan
IBM Research Laboratory
San Jose, CA

Al Hawkes
Sargent & Lundy Engineers
Chicago, IL

Russell K. Brown
Moore Paper Company
Houston, TX

Harvey L. Garner
University of Pennsylvania
Philadelphia, PA

Jerry Koory
On-Line Business Systems, Inc.
Los Angeles, CA

Henry S. McDonald
Bell Laboratories
Murray Hill, NJ

1976 NATIONAL COMPUTER CONFERENCE

Chairman

Carl Hammer
Sperry Univac
Washington, DC

1977 NATIONAL COMPUTER CONFERENCE

Chairman

Portia Isaacson
University of Texas
Richardson, TX

1978 NATIONAL COMPUTER CONFERENCE

Chairman

Stephen W. Miller
Stanford Research Institute
Menlo Park, CA

SESSION CHAIRMEN

Ahl, David H.
Alberts, David S.
Amarel, Saul
Amenta, Joyce A.
Anderson, Walter L.

Baldwin, Lloyd
Berg, John L.
Berglund, Patricia
Betz, Nancy
Boardman, Thomas L.
Bomzer, H. W.
Borgerson, Barry R.
Brainerd, John G.
Branstad, Dennis K.
Brewer, Susan
Brown, Carol
Butler, Margaret
Buzen, Jeffrey P.

Chu, Yaohan
Cochran, Anita J.
Cotton, Ira
Courtot, Marilyn E.

Davis, John C.
Davis, Ruth M.
DeGeorge, Frank D.

Eberlein, Patricia
Estrin, Gerald
Estrin, Thelma
Etra, Louise R.
Evans, B. O.

Farber, David J.
Feigenbaum, Edward A.
Firestone, Roger M.
Fox, Margaret
Frank, Ronald A.
Friedman, Leonard
Fritz, W. Barkley

Gey, Fredric C.
Gildea, Robert
Goldstine, Herman
Green, Teresa O.
Greenwald-Katz, Genevieve
Gregory, Francis M. Jr.
Gregory, Neal
Grinoch, Etelle
Gulick, Anne M.

Hedges, Harry G.
Hirst, Norman F.
Jackson, Peter E.
Jacob, Jean-Paul
Jamison, Steven L.
Johnson, Carol
Joshi, A. K.
Joslin, Edward O.

Kaenel, Reg A.
Kahn, Robert E.
Kameny, Iris
Kapur, Gopal K.
Keller, Roy F.
Ketchel, James S.
Kiehl, Janet
Kimbleton, Stephen R.
Kiviat, Philip J.
Klingman, Darwin
Kuo, Franklin F.

Ledley, Robert S.
Levine, Stephen
Lipsen, Janice C.
Lowenthal, Eugene L.

Macon, Nathaniel
Madden, Thomas J.
Maisel, Herbert
McLeod, John
Mellen, Greg E.
Merten, Alan G.
Merwin, Richard E.
Miller, Edward F. Jr.
Minsky, Marvin
Mishelevich, David J.
Morgan, Howard L.
Moshman, Jack
Murphy, Evelyn R.

Neary, Dennis R.
Nelson, Eldred C.
Oliver, Paul
Oliver, S. Ron

Padwo, Saul
Pantell, Bernice
Papert, Seymour
Patinella, Anthony J.
Perry, William E.
Poh, Susan S.
Poppel, Harvey L.
Potts, Jackie
Pouzin, Louis
Proctor, Bernice J.
Prywes, Noah S.
Pyke, Thomas N. Jr.

Raben, Joseph
Rauscher, Tomlinson G.
Rieger, Charles
Ruspini, Enrique

Sager, Naomi
Salasin, John
Scala, Joseph
Scala, Patsy
Scharpf, Norman W.
Schneck, Paul B.
Sedelow, Walter A. Jr.
Seligman, Naomi
Shepard, Richard H.
Soden, John V.
Speer, Richard
Svobodova, Liba
Swift, Stephen T.

Turn, Rein
Turoff, Murray

Weber, William R.
Welke, Larry A.
Winkler, Stanley

Yau, Stephen S.
Yeh, Raymond T.
Yourdon, Edward
Yovits, Marshall C.

PARTICIPANTS

Adler, J.
Allan, J. J. III
Allen, D.
Allen, J.
Amelio, G. F.
Anderson, D. K.
Anderson, J. P.
Anderson, P. F.
Anderson, P.
Anderson, R.
Armer, P.

Backer, D.
Bacus, J. W.
Baecker, R. M.
Baird, G. N.
Baker, T.
Baldwin, R. C.
Bales, H.
Balkovich, E.
Balzer, R.
Barefoot, D. B.
Barlow, J. S.
Barnett, G. O.
Baumberger, B. E.
Bearden, G. D.
Beeler, M.
Bell, R.
Bell, T. E. Jr.
Bendick, M.
Benton, J. B.
Bigelow, R. P.
Boehm, B. W.
Boggs, D. R.
Bookwalter, T.
Bork, A.
Bowie, J.
Branum, A.
Bridger, D. A.
Broos, M. S.
Brown, C. L.
Brown, J. S.
Brown, R.
Buchanan, B.
Budzik, P.
Burchfiel, J.
Burntyk, N.
Burrows, J. H.
Bushkin, A. A.

Carabello, J. M.
Carter, G.
Chaitin, G.
Chankter, A.
Chapman, R.
Chapman, W. E. III
Chen, P.
Chmura, J.
Cho, Z. H.
Chow, W.
Chu, C.
Clay, B. M.
Clippinger, R.
Cohen, K. J.
Collins, J.
Congdon, M.
Conners, R. W.
Constable, T.
Constantine, L.
Cook, M. M.
Cornish, L.
Cosloy, E. S.
Cossette, A.
Cremer, R.
Crissey, B. L.
Cruver, H. F.
Custer, D. D.

Darby, L.
Day, J.
Deese, D. R.
DeGeorge, F. D.
Denning, P. J.
Dennis, J.
Dodd, G. G.
Dooling, L.
Dorodnicin, A. A.
Dunion, J.
Durk, D.
Dyer, S. J. III

Eastman, C. M.
Eichert, E. S.
Ellis, C.
Engel, F. Jr.
Engelbart, D.
Epperly, E. V.
Erickson, R. F.
Erman, L.
Etra, W.
Eulenberg, J.

Fields, C.
Figer, J. P.
Foulds, R.
Frampton, L.
Frankfeldt, C.
Franklin, M.
Franklin, W.
Freed, R. N.
Furtman, E. L.
Fuss, D.

Gagliardi, U.
Gardner, R. I.
Gearhart, J. B.
Gill, J. M.
Giorgini, A.
Glaser, G.
Gluck, S. E.
Gluckson, F.
Goetz, M. A.
Goldberg, B.
Goldberg, J.
Goldberg, P.
Goldfield, R. J.
Goldstein, C. M.
Goldstein, I.
Golub, H.
Goodenough, J. B.
Graham, M.
Graham, R. M.
Gray, H. J.
Greenberg, D.
Gruenberger, F.
Guiltinan, R. J.

Hagstrom, S.
Hall, T.
Hanna, W. E. Jr.
Harder, D. C.
Harlow, C. A.
Harvey, F. L.
Hawrylyshyn, P.
Heaton, D.
Heil, S. W.
Hendler, H.
Henriquez, V.
Herman, G. T.
Herrmann, W. T.
Higgs, L. D.
Highland, H. J.
Hirschsohn, I.
Hobbs, J. R.
Holloway, C. A.
Holthouse, M. A.
Housel, B. C.
Hoylman, J. M.
Howard, P. C.
Howard, S.
Hoyt, P. M.
Hubbert, D. S.
Hutton, W. E.

Isaacson, P.
Ivaldi, A. M.

Jackson, M.
Jacobs, I.
Janiczek, P. M.
Jasper, D. P.
Jelinek, F.
Jenkin, M. A.
Johnson, Cecil
Johnson, E.
Johnson, E. C.
Johnson, H. A.
Johnson, R. R.
Jones, C.
Joseph, E. C.
Justice, N.

Kalfel, J.
Kanodia, R. K.
Kaplan, A. R.
Karplus, W. J.
Kashmari, S. A.
Katz, L.
Kay, A.
Kennington, J.
Kent, C.
King, D.
King, J. C.
Kirkley, J. L.
Kleiman, H.
Knowlton, K.
Kroneberg, D.
Kudo, T.
Kulikowski, C.
Kurzweil, R.

LaDue, R. B.
Lansberry, C. R.
Larsen, G. N.
Larson, A. G.
Laufer, R.
Leary, W.
Lee, R. M.
Leffler, L. C.
Lerner, L.
Lester, A. L.
Lipkin, L. E.
Liskov, B.
Little, J.
Lodwick, G. S.
Longsworth, M. A.
Lubin, J. F.
Lukoff, H.
Lum, V. Y.
Lundell, E. D.
Lyet, J. P.

Malone, W.
Mallary, R.
Mandell, M. S.
Marcus, M.
Margeson, A. J.
Marion, R. A.
Marks, S.
Martin, S.
Martin, T.
Mason, P. H.
Mathews, M. V.
Mauchly, J.
Mayhew, W.
McCluskey, E. J.
McDonald, D.
McGowan, C.
McLean, J.
McNeal, J. K.
McRae, W. B.
Meehan, J.
Meissner, L. P.
Mercer, R. L.
Metcalfe, R. M.
Meurer, T.
Mills, H. D.
Moldow, B.
Morino, M. M.
Morris, J. A.
Mullinax, J.
Myer, T.
Myers, G. J.

Nash-Webber, B.
Nelson, T. H.
Nemeth, A.
Ness, D.
Nordlund, D. L.
Normark, W. F.
Norton, H.
Nunamaker, J. F.

Oestreicher, D.
Ogdin, J.
O'Leary, H. E. Jr.
Omata, S.
O'Neill, J. T.
Opderbeck, H.
Organ, L. W.

Page, L. F.
Palacios-Hammeken, L.
Palmer, F. B.
Patrick, P. H.
Patton, S. K.
Picard, P.
Pohl, F.
Pollock, K. A.
Poor, V. D.
Pople, H.
Postle, W.
Potash, H.
Quake, R. P.
Quong, C.

Ragland, T. R.
Ralston, A.
Randall, G.
Rather, E. D.
Rattner, J.
Reifer, D. J.
Rettberg, R.
Ries, D.
Ritchie, D. M.
Ritea, B.
Roark, M. L.
Robertson, T. A.
Rose, C.
Rosen, S.
Rosenschein, S.
Rosin, R. F.
Ross, D. T.
Rowe, I. H.
Rule, D.
Rulifson, J.
Russell, R. D.

Sanchez, E.
Sanchez, W.
Sanders, R.
Sandin, D.
Savides, P.
Schmedel, S. R.
Schmidt, C.
Schmidt, S. C.
Schmidt, W. B.
Schneidewind, N. F.
Schneidman, A.
Schultz, M.
Schwartz, J.
See, M.
Selfridge, O.
Selig, S. M.
Sell, R.
Shair, H.
Shapiro, S.
Shay, B. P.
Sherman, A. H.
Shortliffe, E. H.
Shourt, J.
Shu, N.
Simmons, R. B.
Slutzky, J.
Smaby, R.
Smith, R. E.
Smith, W. H.
Snyder, J. N.
Souder, D. E.
Spiegel, L.
Sridharan, N. S.

Steele, T. Jr.
Stoian, E. R.
Stokes, G.
Stout, H. S.
Stazisar, V.
Strecker, W. D.
Sutherland, I.
Suzuki, R.
Szolovits, P.

Tajima, K.
Takase, H.
Tasker, R. R.
Taulbee, O. E.
Teicholtz, E. D.
Tillinghast, J.
Thayer, T.
Thomas, K. L.
Thompson, G. L.
Tikson, M.
Tomlinson, R.
Travis, I.
Tretiak, O.
Tucker, A. Jr.
Tunnicliffe, W. W.

Uhlig, R. P.
Uzgalia, R. C.

Van Natta, B.
Vaughan, W. K.
Vezza, A.
Vidal, J.
Volz, R. A.

Wade, B. W.
Walden, D.
Walker, D. L.
Walker, D.
Walker, P. D.
Walker, S. T.
Walrod, R. A.
Walters, R. F.
Waltz, D.
Watkins, W.
Weber, R. A.
Wegner, P.
Wein, M.
Weller, M.
Wells, W. I.
Westin, A. F.
White, H. S.
White, M.
Whitney, R.
Whittaker, W.
Wilkes, M.
Williamson, H.
Wilson, M. E.
Winston, J. S.
Withington, F.
Wolf, E. W.
Wolff, A.
Wood, R.
Wray, W.
Wright, H.
Wright, L. S.

Yakimovsky, Y.

Zelkowitz, M.
Zingg, R. J.

NCC '76 REFEREES

Abbey, Mary W.
Abrams, Marshall D.
Ackerman, L. V.
Adams, E. N.
Adams, Elizabeth Byrne
Adams, J. Alan
Adams, Scott
Adams, William A.
Agrawala, Ashok K.
Aicher, J. R.
Aiken, Robert M.
Aines, Andrew A.
Airhart, T. E.
Albers, Glen
Aldred, William M.
Alexiou, John K.
Ali, Phi
Allan, John J. III
Allen, John R.
Allen, Richard D.
Allen, Rodney H.
Alpert, Stephen R.
Alter, Ralph
Amarel, Saul
Amicone, Ray C.
Andersen, Niels C.
Anderson, Peter G.
Anderson, Robert H.
Anderson, Sherwood E.
Anderson, T. C.
Andree, R. V.
Archer, David A.
Archibald, Julius A. Jr.
Armer, Paul
Arnovick, G. N.
Aron, J. D.
Arterbery, Vivian J.
Ash, Alvin
Asprey, Winifred
Astrahan, M. M.
Atchison, William F.
Atherton, Pauline
Atkins, D. E.
Atwood, Delbert W. Jr.
Augustin, D. C.
Aupperle, Eric M.
Austin, Donald M.
Austing, Richard H.
Avrunin, Ira L.
Ayer, Nancy L.
Ayers, Lawrence F.

Bachman, Charles W.
Baer, J. L.
Bagley, James E.
Bahn, Michael M.
Bailey, Eugene
Baird, George
Baker, F. T.
Baker, James A.
Baker, Lara H. Jr.
Baker, R. A.
Baker, Robert L.
Ball, N. Addison
Baltzer, Philip K.
Balzer, Robert M.
Bandurski, Ann Ellis
Banerji, Ranan
Barnes, Bruce H.
Barnes, Robert F. Jr.
Barnett, G. Octo
Barnett, Robert T. Jr.
Barr, Avron
Barr, William J.
Barrett, William A.
Basili, Victor R.
Bassler, Richard A.
Bate, Roger R.
Bateman, Barry L.
Bauman, Burton L.
Bayer, Gary E.
Beall, William H.
Bearden, G. D.
Beaton, Albert E.
Beck, Leland
Beck, Paul N.
Bednar, Gregory M.
Bein, Donald H.
Beitz, E. Henry
Belady, L. A.
Belden, Glen W.
Belford, Geneva
Bell, Thomas E.
Bellan, T. M.
Belzer, Jack
Bemer, Robert W.
Bennett, John
Benson, Walter R.
Berg, John L.
Berger, Ralph
Berger, Ray
Berglund, Ralph G.
Berk, Toby
Berning, Paul T.
Bernstein, George B.
Bernstein, M. I.
Bernstein, Ralph
Berra, Bruce P.
Bertaut, Edgard F.
Bewley, William L.
Bigelow, Robert P.
Billingsley, Fred C.
Binder, Richard
Binford, Thomas O.
Bingham, Harvey W.
Bise, Robert G.
Bitterli, Charles V.
Bitz, Ira
Bjorner, Dines
Black, Donald V.
Blanc, Robert P.
Blomgren, George H.
Bloomfield, James A.
Blue, Richard B. Sr.
Blum, Joseph
Bodoia, Morris J.
Bollenbacher, Roger L.
Bolton, Ronald M.
Bono, Peter R.
Booth, Grayce M.
Booth, Taylor L.
Borgerson, Barry R.
Bork, Alfred
Borko, Harold
Bouknight, Jack
Bourne, John R.
Brackett, John W.
Bradshaw, Charles L.
Braithwaite, Timothy
Brandejs, Jan F.
Brandon, Daniel M. Jr.
Branscomb, Lewis M.
Branstad, Dennis K.
Bratman, Harvey
Bremer, John W.
Brennan, R. D.
Bressler, Robert
Brickman, Norman F.
Bright, Herbert S.
Brignoli, Frank
Brociner, Betty B.
Brown, John R.
Browne, J. C.
Browne, Peter S.
Bruce, Bertram
Bryan, G. Edward
Bucher, Michael D.
Burns, Joseph L.
Burns, S. K.
Burton, Hilary
Burton, William D. Jr.
Buscher, David J.
Butler, George A.
Butler, Robert
Butterfield, Max A.

Cady, George M.
Callahan, J. A.
Campaigne, H.
Campbell, John B.
Campise, James A.
Cannon, Robert L. Jr.
Caplan, D.
Capsis, George
Cardenas, Alfonso F.
Cardwell, David W.
Carey, Bernard J.
Carlson, Carl R.
Carlson, Eric D.
Carlson, Gary
Carlson, Richard R.
Carmichael, Bruce
Carmichael, Robert L.
Carr, John W. III
Carroll, Allen M.
Carter, George
Case, James A.
Case, Leon R. II
Case, Richard P.
Casey, Richard G.
Cashman, Thomas J.
Cashton, Sidney
Castruccio, Peter A.
Cave, Randal H.
Cea, Eugene J.
Chamberlin, Donald D.
Champine, G. A.
Chan, Maynard M. W.
Chang, H. Y.
Chang, Hsu
Chao, Yen W.
Charp, Sylvia
Chase, Harlan C.
Chauhan, Rohi
Cheatham, Thomas E.
Chen, Peter
Chen, Robert C.
Chen, Thomas T.
Chen, Tien Chi
Chernak, Jess
Cheung, Roger C.
Cheydleur, Ben F.
Chi, Emile C.
Chinitz, M. P.
Cho, Seon H.
Chou, Wushow
Chu, W. W.
Chu, Yaohan
Clark, David D.
Clark, Stephen C. III
Climis, Ted E.
Clymer, James R. W.
Cochran, Anita
Codd, E. F.
Cody, William J. Jr.
Cohen, Dan
Cohen, Jack
Cole, G. D.
Coleman, Michael L.
Coles, L. Stephen
Collmeyer, Arthur J.
Colub, Gene H.
Condon, S. F.
Conner, William M.
Cook, Gord
Cook, Meyer
Cook, Peter G.
Cooprider, Lee W.
Corduan, Alfred E.
Corley, Melvin R.
Cornish, Edward S.
Cotterman, William W.
Cotton, Ira W.
Cottrell, Norman E.
Couch, John Dennis
Couger, J. Daniel
Couperus, J.
Courtright, Benjamin F.
Cowan, Robert J.
Cowgill, Daniel E.
Creel, R. E.
Creveling, Cyrus J.
Crocker, Dean D.
Culpepper, L. M.
Curtis, Kent K.

Dacey, Michael F.
Dahm, David M.
Dalenius, Tore
Dalphin, John F.
Daniels, E. L.
Daniels, Walter E. Jr.
Danner, Lee
Davida, George
Davidson, Donald A.
Davies, W. Ronald
Davis, Alan
Davis, C. M.
Davis, Carl G.
Davis, John C.
Davis, R. M.
Day, Paul
Deal, Richard L.
Dean, Edwin B.
Deb, Rajat K.
De Bons, Anthony
Defiore, Casper R.
De Greene, Kenyon B.
De Regt, Maurits P.
Devine, Edward P.
Dickson, Charles H.
Diethelm, M. A.
Dixon, Louis F.
Dobkin, David
Dockery, John T.
Dodd, George
Donaldson, Fletcher W.
Dorn, Philip H.
Douglas, John R.
Douthat, Dean Z.
Dowling, Terry
Drane, Douglas
Draper, George L.
Drattell, Alan
Duane, Darrell W.
Ducasse, Edgar
Dumey, Arnold I.
Duncan, Karen
Dunlavey, Richard
Dunn, Robert M.
Dutka, Jacques
Dwyer, Samuel J. III
Dylewski, T. J.
Dym, Charles H.

Easley, Joseph H.
Eastman, Charles
Eccles, William J.
Eckhouse, Richard H. Jr.
Edwards, Judith B.
Edwards, W. Allan
Ehlers, Marvin W.
Eirich, Donald L.
Eirich, Peter L.
Eisen, Lawrence
Elfant, Robert F.
Elkins, Bryce L.
Elliott, David W.
Elliott, Glenn R.
Elman, Stanley A.
Emerson, E. James
Engel, Diana
Engel, Gerald L.
Enslow, Philip H. Jr.
Erickson, Raymond F.
Ernst, George W.
Esch, John W.
Estock, Richard G.
Estrin, Thelma
Evans, W. Buell

Fabry, R. S.
Faiman, M.
Farmer, Nick A.
Farmer, Victor J.
Faust, Hilda C.
Fein, Alvin E.
Feingold, Robert S.
Feldman, I.
Felton, Walter W.

Feng, Tse-Yun
Ferrari, D.
Feurzeig, Wallace
Feustel, Edward Alvin
Feyock, Stefan
Fife, Dennis W.
Finerman, Aaron
Firestone, Roger M.
Firschein, Oscar
Fischler, Martin A.
Fisher, Charles
Fisher, David A.
Fisher, Donald D.
Fisher, Gerald A. Jr.
Fisher, John M.
Flood, Merrill M.
Fogel, Marc H.
Foley, James David
Fong, Elizabeth
Foster, C. L.
Foster, Caxton C.
Foster, David F.
Fowler, Bruce
Fowler, Mary
Fox, C. Robert
Fox, Margaret R.
Fox, Phillip W.
Frailey, Dennis J.
Frank, Howard
Frank, Ronald A.
Frank, Werner L.
Franke, Richard
Franklin, Jeff
Fredman, Irwin J.
Fredman, Michael L.
Freeman, Martin
Freiman, C. V.
French, L. J.
Friedman, Daniel P.
Friedman, Jerome H.
Friedman, Lee A.
Fritsch, John M.
Fritz, W. Barkley
Fruchter, Murray
Fujiwara, Harry A.
Fuller, Richard H.
Futrelle, Robert

Gagliardi, Ugo O.
Galler, Bernard A.
Gallo, Arpaol
Gammill, Robert C.
Gammon, William Howard
Gangl, Erwin C.
Gannon, John D.
Gantner, George E.
Gardner, Jeffrey J.
Gardner, Willard H.
Garrett, Richard E.
Gass, Saul I.
Gates, G. W.
Gates, Roy
Gaylord, C. V.
Gear, C. W.
Gerla, Mario
Gey, Fredric C.
Gibb, Kenneth R.
Giesa, C. Eric
Gilchrist, B.
Gilliland, B. E.
Gimpac, James F.
Gitman, Israel
Glanc, Alois
Glaseman, Steve
Glaser, George
Glasser, Robert G.
Gleissner, Gene H.
Glick, Norman S.
Glorioso, Robert M.
Goetz, Donald F.
Goetz, Martin A.
Gold, Archie
Goldberg, Adele
Goldberg, Jack
Goldhirsh, Isodore L.
Golding, E. I.
Goldman, Neil M.
Goldstein, Aaron
Goldstein, Charles M.
Gonzalez, Mario J. Jr.
Goodman, Arnold
Goodrich, Roger E.
Gorgone, John T.
Gorman, Donald F.
Gorman, Michael M.
Gorschboth, F. F.
Gorsline, G. W.
Gosden, J.
Gose, Anne Hamilton
Goshen, Robert J.
Gould, John D.
Gowdy, John N.
Grace, Alonzo G. Jr.
Graham, G. Scott
Gralia, Mars J.
Grampp, F. T.
Grau, Albert A.
Gray, James N.
Greaves, John O. B.
Green, Duff III
Green, Teresa O.
Greenawalt, E. M.
Greenes, Robert A.
Griffin, John R.
Grimes, Dale M.
Grishman, Ralph
Grobstein, David L.
Groner, Gabriel F.
Groner, Leo H.
Grossman, George
Grossman, H. B.
Gruhn, Ann M.
Guetzkow, Harold
Guiteras, Joseph J.
Gump, Raymond D.
Gussow, Milton
Gyllstrom, Hans C.

Habermann, A. N.
Habib, Stanley
Hadjioannou, Michael
Hall, Wayne A.
Hamblen, John W.
Hamilton, Dennis E.
Hamilton, William A.
Hamlet, Richard G.
Hammer, Fred E.
Hammer, Michael
Hammer, Preston C.
Hamming, R. W.
Hampel, D.
Hampson, Richard K.
Hanna, William E. Jr.
Hansen, John C.
Hantler, S. L.
Hardgrave, W. T.
Hardy, Ann
Harlaw, Charles
Harmon, John B.
Harold, Frederick G.
Harper, Jackson D.
Harrell, Clayton Jr.
Harris, Daniel K.
Harris, Floyd O.
Harris, Fred H.
Harris, Richard D.
Harris, William I.
Harrison, Thomas J.
Harstad, Kenneth R.
Harter, M. D.
Hartford, Donald L.
Hartley, Dean S. III
Hartwick, R. Dean
Hattery, Lowell H.
Hays, Bill
Hays, David G.
Head, Robert V.
Heart, Frank E.
Hedrick, George E.
Heintz, Alden
Helgeson, Duane M.
Heller, Jack
Hellwarth, George A.
Henderson, D. Austin Jr.

Hennessey, James F.
Henriques, Vico E.
Henschen, Lawrence J.
Hernon, James A.
Hertlein, Grace C.
Hess, George J.
Heying, Douglas W.
Higgins, A. N.
Higgins, Thomas J.
Highland, Harold Joseph
Hill, Fredrick, J.
Hill, Harold E.
Hillman, Donald J.
Hinomoto, Hide
Hirsh, Cathy
Hixson, Harold
Ho, Thomas I. M.
Hobbs, Jerry R.
Hodge, Thea D.
Hodson, Richard B.
Hoffman, John M.
Hoffman, Lance J.
Hoffman, R. H.
Hofler, Richard D.
Hohn, Franz E.
Holden, Alistair D. C.
Holden, Willard J.
Hollander, G. L.
Hollingworth, Dennis
Hollist, William Ladd
Holme, Dorothea
Holmes, Harvard
Home, William J.
Hook, H. O.
Hoover, L. Ronald
Hopewell, Lynn
Hopper, Grace M.
Hopwood, Gregory L.
Hopwood, Marsha D.
Hord, R. Michael
Horne, William J.
Hoskins, W. D.
Houston, H. Richard
Howard, John H. Jr.
Howell, Jo Ann
Howell, Thomas H.
Howerton, Paul W.
Howley, Frank E.
Hoyt, Patrick M.
Hsiao, David K.
Huang, H. K.
Huckell, Gary R.
Hughes, Charles E.
Hughes, James
Hunt, Hurshell H.
Huntwork, Paul K.
Hutchison, John S.
Hutt, Arthur

Hwang, Frank K.

Iberall, Arthur
Ingerman, Peter Zilahy

Jacobi, George T.
Jacobs, Howard
Jacobs, Walter
Jacobus, G. C.
James, Thomas A.
Janac, Karel
Jarvis, John F.
Jefferson, David K.
Jeffery, S.
Jensen, Alton P.
Jensen, Raymond A.
Jercinovic, L. M.
Jessep, Donald C. Jr.
Johnson, A. I.
Johnson, Carl W.
Johnson, David L.
Johnson, Walter A.
Johnson, Walter L.
Jones, Alice U.
Jones, Anita K.
Jones, Jack
Jones, Neil D.
Jones, Ronald Dale
Jones, William J.
Jordan, Thomas L. Jr.
Joseph, Earl C.
Joshi, A. K.
Joslin, Edward O.
Joyce, Charles C. Jr.
Joyce, J.
Joyner, William H. Jr.

Kaenel, R. A.
Kafafian, Haig
Kagan, Claude A. R.
Kahng, S. W.
Kain, Richard Y.
Kampen, Garry R.
Kanal, Laveen
Kandel, Abraham
Karplus, Walter J.
Kasarda, Andrew J.
Kaster, Charles G.
Katzper, Meyer
Kay, Ira M.
Kazek, Chester S. Jr.
Keller, Robert M.
Keller, Roy F.
Kennevan, Walter J.
Kieburtz, Richard B.
Kimbleton, Stephen R.
King, James C.
King, Willis K.

Kirshenbaum, Frank
Kiviat, Philip J.
Klassen, Daniel L.
Klee, Otmar A.
Kleinrock, Leonard
Kleitman, Daniel J.
Klerer, Melvin
Knight, Douglas Wayne
Knight, J. C.
Knowlton, Prentiss
Koch, Harvey S.
Kornfield, N. R.
Koss, Adele Mildred
Kovach, Ladis D.
Koymen, Kemal
Kozik, Eugene
Kraley, Michael F.
Kraska, Paul W.
Krause, Kurth
Kretchmar, A. L.
Kroeger, Joseph H.
Kronenberg, Nancy
Krueger, E. Rex
Krulee, Gilbert K.
Krummel, Larry
Kuch, T. D. C.
Kuo, Franklin F.
Kurtzberg, Jerome M.

LaFrance, Jacques E.
Laliotis, Ted
Lam, Simon S.
Lambert, Robert J.
Lamothe, Ray J.
Landau, Robert
Lane, Malcolm G.
Larson, Arvid G.
Lasser, Daniel J.
Latker, Alex C.
Laurance, Neal
Lawrie, D.
Lazar, Leonard M.
Leasure, Bruce R.
Leavitt, Michael R.
Le Beux, Pierre J.
Ledin, Victor
Ledley, Robert S.
Lee, Jan
Lee, Marshall
Lee, Robert M.
Lee, Samuel C.
Lennon, James J.
Lesk, Michael
Lester, Bruce P.
Levin, Roy
Lew, Art
Lien, Y. Edmund
Lillestrand, R. L.

Lin, Shen
Lin, Wen T. K.
Lincoln, A. James
Lincoln, Neil R.
Linde, Richard R.
Linden, Theodore A.
Lindsay, Bruce
Link, C. H.
Lipow, Myron
Lippman, Mike
Liskov, Barbara
Littrell, R. F.
Liu, C. L.
Liu, Jane W. S.
Lividini, Joseph
Lloyd, Jack
Logan, J. J.
Logue, Joseph C.
Lomet, David B.
Long, Harvey S.
Loomis, Donald C.
Lorie, Raymond A.
Lovegrove, D. H.
Lowe, Thomas C.
Lozier, Daniel W.
Lucas, Henry C. Jr.
Lucido, Anthony P.
Luck, Dennis R.
Luderer, Gottfried W. R.
Ludwig, Herbert R.
Luk, Clement
Lukas, George
Lukoff, Herman
Lum, Vincent Y.
Lumb, Arthur C.
Lundell, E. Drake Jr.
Lurie, Dan
Lutz, Michael J.
Luxenberg, H. R.
Lycklama, H.
Lykos, Peter
Lyle, Robert F.
Lynch, John T.
Lyon, M. S.
Lyons, Robert E.
Lyons, W. W.

Macaleer, R. James
Machover, Carl
Madnick, Stuart
Madrigal, Orlando S.
Madron, Beverly B.
Maguire, John N.
Mahoney, Michael
Mahoney, William C.
Maisel, Herbert
Maish, Alexander M.
Mamrak, Sandra
Mandell, Steven L.
Maniotes, John
Mann, Richard L.
Manola, Frank
Manuel, Thomas J.
Maple, Clair G.
Marcantonio, Angelo R.
Marcovitz, Alan B.
Marmor-Squires, Ann
Marrigan, Robert J.
Marsland, T. A.
Martino, Joseph P.
Martins, Gary R.
Matheny, Charles S.
Mathews, M. V.
Mathews, Walter M.
Mathis, Charles L.
Mathison, Stuart
Mathur, F. P.
Matyas, Stephen M.
Mc Carn, Davis B.
Mc Carthy, John F. Jr.
Mc Clean, R. K.
Mc Cluskey, E. J.
Mc Connell, Thomas J. Jr.
Mc Cready, R. R.
Mc Cuskey, William A.
Mc Daniel, Herman
Mc Donald, Clement
Mc Donald, James C.
Mc Fadden, Ted
Mc Gill, Michael J.
Mc Gregor, P. V.
Mc Ilroy, M. D.
Mc Jones, Paul
Mc Kenna, James K. Jr.
Mc Knight, Randy S.
Mc Leod, Dennis J.
Mc Murran, M. W.
Meads, Jon A.
Mee, Carl III
Mehl, James W.
Meissner, Loren P.
Melkanoff, M. A.
Mellen, Greg
Meltzer, Herb
Menard, John P.
Mentges, Charles W.
Merwin, Richard E.
Metcalfe, Robert M.
Metzner, John R.
Michel, Martin J.
Mihram, G. A.
Miles, E. P. Jr.
Milgram, David L.
Miller, Jack M.
Miller, Lawrence H.
Million, E. Z.
Mills, David L.
Mills, Harlan D.
Milner, Stuart
Mink, Thomas A.
Minker, Jack
Minsky, Naftaly
Mintz, Daniel
Misek, L. D.
Mishelevich, David J.
Mitchell, Terry R.
Modesitt, Kenneth L.
Moik, Johannes G.
Moler, Cleve
Moraff, Howard
Morgan, M. Granger
Morgenstern, George
Morris, Joel
Morris, Michael F.
Morton, A. Kent
Morton, Michael S. Scott
Moshman, Jack
Moshos, G. J.
Moyles, Dennis M.
Mucciardi, Anthony N.
Muchnick, Steven S.
Muhlhauser, Robert R.
Mullany, James
Mullen, Karen A.
Mulroney, William C.
Murray, Thomas E.
Muzio, J. C.

Nagel, Roger
Nance, Richard E.
Nasem, Charles
Nash-Webber, Bonnie
Nee, David S.
Neely, Peter M.
Negroponte, Nicholas
Nelson, Eldred
Nemeth, Alan G.
Neuenschwander, Charles R.
Neurath, Peter W.
Nevins, J. L.
Newman, Shelley
Newton, Carol
Nichols, A. J.
Niedermair, F. Robert
Nielson, Gregory M.
Nielsen, William C.
Nievergelt, J.
Noetzel, Andrew S.
Noonan, Robert E.
Norman, Theodore A.
Notestine, R. E.
Nugent, William R.
Nutt, Gary J.
Nuxall, John W.

Oberg, James E.
O'Kane, Kevin C.
Oliver, Paul
Oliver, S. Ron
Olmer, Jane
O'Neill, Dennis M.
Osher, William J.
Ossanna, Joseph
Osterweil, Leon
Otolle, James A.
Overstreet, Claude
Owens, John D.

Palermo, Frank P.
Palley, Norman
Palmer, Richard
Palting, Cezarina A.
Parish, Randall M.
Parker, Donn B.
Parke, Benjamin G.
Patrick, Edward A.
Patterson, James W.
Patton, S. K.
Patt, Yale N.
Payne, Mary
Pearson, Karl M. Jr.
Pease, M. C.
Peavey, Ross D.
Peck, Larry J.
Pehlert, William K.
Penderghast, Thomas F.
Perlis, Alan J.
Perry, Raymond S.
Perry, William E.
Peters, Carol B.
Peterson, Emery G.
Peterson, James
Peterson, Tom
Pfleeger, Charles
Phillips, Charles A.
Pickholtz, Raymond L.
Pinson, E. N.
Pipberger, H. V.
Pirtle, Mel
Pitts, Gerald N.
Pizer, Stephen M.
Plauger, P. J.
Plourde, Paul J.
Podell, Harold J.
Pool, Ithiel
Poole, Peter C.
Poore, Jesse H. Jr.
Popek, G. J.
Popino, J. P.
Potter, Marshall R.
Potts, Jackie
Pouring, A. A.
Powers, Ernest F.
Powers, Richard
Pratt, Arnold W.
Prescott, Lee R.
Press, Barry
Price, Robert
Prokop, Jan
Pugsley, Ronald S.
Purdy, J. Gerry
Pyke, T. N. Jr.

Quann, John J.
Quinlan, Chris R.

Raben, Jeffrey M.
Raben, Joseph
Rabinow, Jacob
Rabinowitz, Irving N.
Radwin, Mark S.
Raichelson, Gene
Raisig, Paul J. Jr.
Rajchman, Jan A.
Raj-Karne, D. G.
Rakoczi, Laszlo L.
Ramamoorthy, C. V.
Ramsay, W. Bruce
Rauscher, Tomlinson G.
Reddi, S. S.
Redell, David D.
Reifer, Donald J.
Reisner, Phyllis
Reiss, R. A.
Reitman, Julian
Reynolds, Brian M.
Rheinboldt, W.
Rice, John R.
Rich, Robert P.
Richardson, Dana Roland
Richardson, Duane E.
Riddle, William E.
Rieger, Charles J. III
Rigg, George P.
Rigney, Joseph W.
Riley, Winston III
Risse, Joseph A.
Ritea, H. Barry
Rittersbach, George H.
Rizza, J. B.
Rizzi, Anthony M.
Roberts, Alfred E.
Robinson, John
Robinson, Lucian
Rockart, John F.
Rodriguez-Rosell, Juan
Rogers, David F.
Rohr, John A.
Ronayne, Maurice F.
Rose, Lawrence L.
Roseman, Jack
Rosen, Robert
Rosenbaum, Susan L.
Rosenfeld, Azriel
Rosenthal, Charles W.
Rosin, Robert F.
Ross, Douglas T.
Roth, R. Waldo
Rothenbuecher, Oscar H.
Rotolo, Louis S.
Rubey, Raymond J.
Rubin, Arthur I.
Rudberg, Donald A.
Ruh, Lawrence A.
Russo, Paul M.
Ruth, Stephen
Ruthberg, Zella G.

Safford, Herbert B.
Sager, Gary R.
Sager, Naomi
Salasin, John
Salton, G.
Saltzberg, Bernard
Salz, Fred R.
Salzman, Roy M.
Samek, Michael J.
Samson, Thomas F.
Sarahan, B. L.
Sashkin, Lawrence
Sassenfeld, Helmut M.
Savage, John E.
Savage, Patric
Scanlon, J. M.
Schaffner, Mario R.
Scharff, Leon
Scher, Julian M.
Scherr, Allan L.
Schiffman, R. L.
Schlegel, C. T.
Schmidt, W. G.
Schmidt, William P.
Schneck, Paul B.
Schroeder, Michael D.
Schroeder, Thomas V.
Schubert, L. K.
Schultz, Gaymond W.
Schumacker, David
Schumacker, R. A.
Schuster, Stewart A.
Schutzer, Daniel
Schwartz, James A.
Schwartz, Mischa
Schwomeyer, Warren A.
Scott, James L.
Scott, Robert H.
Scribner, Paul
Scrutchin, Thomas W.
Seals, Eugene

Sedelow, Sally Yeates
Seidel, Robert J.
Sekino, Warren T.
Sevcik, K. C.
Shahbender, Rabah
Shahin, Gordon T.
Shapiro, Sidney
Shapiro, Stuart C.
Shaw, Alan C.
Shaw, Mary M.
Sheffield, Charles
Shell, Donald
Shelly, Gary B.
Shelter, A. C.
Shemer, Jack
Shepley, D. J.
Sherman, Stephen W.
Sherrod, John
Shipman, David L.
Shipton, Harold W.
Shneiderman, Ben
Shoaf, G. H.
Shoemaker, Loren
Shohara, Sei
Short, Gerald E.
Shreckengost, R. C.
Shuey, Richard L.
Sibley, E. H.
Sibley, W. L.
Sickel, Sharon
Siegel, A.
Silvern, Leonard C.
Simmons, Dick B.
Simmons, Edward J. Jr.
Simonyi, Charles
Skeel, Robert
Slamecka, Vladimir
Slaughter, Lawrence H.
Slegel, Raymond C.
Sloan, Martha E.
Smith, Cecil L.
Smith, Clarence O.
Smith, Diane C. P.
Smith, Eugene B.
Smith, F. Gordon
Smith, John Miles
Smith, Nicholas M.
Smith, Richard
Smith, Robert L. Jr.
Smoot, Oliver R.
Sokol, George M.
Sondak, Norman E.
Spacek, Thomas R.
Sparrow, Charles A.
Spier, Michael J.
Spinrad, Robert
Spiro, Bruce E.
Spitzer, Frank
Springe, F. W.
Squires, Stephen L.
Srodawa, Ronald J.
Stahl, Fred
Stallings, William
Stanford, W. Donald
Stanton, J. M.
Steel, Thomas B. Jr.
Stefferud, Einar
Stelmack, Frank
Stemple, David W.
Stevens, David F.
Stoffel, James C.
Stokes, Gordon E.
Stokes, Richard A.
Stone, David L.
Stone, Robert L.
Stonebraker, Michael
Story, James R.
Stout, Curtis M.
Strassmann, P. A.
Strauss, Charles M.
Stroud, W. G.
Strout, Frederick D.
Stuebing, H. G.
Stumpf, Werner E.
Stutzman, Leroy F.
Summit, Roger K.
Suszynski, N.
Sutherland, Ivan E.
Swanson, A. Kenneth
Swenson, J. Richard
Swigger, B. K.
Szygenda, S. A.

Taggart, William M. Jr.
Tam, Wing
Tanaka, R. I.
Tannenbaum, B.
Tarbell, Lawrence C. Jr.
Tartar, John
Taulbee, Orrin E.
Tausner, Miriam R.
Taylor, Robert E.
Taylor, Robert W.
Teichroew, Daniel
Tempas, Jon M.
Tenenbaum, Jay
Tesler, Lawrence G.
Thayer, Richard H.
Theilheimer, Feodor
Thomas, Charles R.
Thomas, Duane J.
Thomas, John C. Jr.
Thomas, Robert H.
Thompson, Howard K. Jr.
Thompson, Ken
Thompson, Martin
Thomsen, D. L. Jr.
Thornton, M. Zane
Tick, Leo J.
Timmreck, Eric M.
Tischhauser, J. L.
Tobagi, Fouad A.
Tonik, Albert B.
Towne, Douglas M.
Townsend, Frederick W.
Travis, Friason G.
Treu, Siegfried
Tucker, Allen
Tucker, Dorothy I.
Tucker, Edwin K.
Tuel, William G. Jr.
Tung, Chin
Turn, Rein
Turner, Jon Anthony
Turoff, Murray

Uhlig, Ronald P.
Uhr, Leonard
Ulery, Dana L.
Ullman, Jeffrey D.
Umpleby, Stuart A.
Utley, E. H.

Valentine, Sol W.
Vallbona, Carlos
Van Dam, Andries
Vanderbrug, Gordon J.
Van Houweling, Douglas E.
Van Slyke, R. M.
Vena, Peter A.
Verry, William
Vick, Charles R.
Vickers, Frank D.
Vierra, Joe C.
Viste, G. D.
Von Almen, Karl

Wachal, Robert S.
Wagner, Charles R.
Wagner, David Jon
Wagner, F. V.
Waite, William M.
Wakerly, John F.
Walden, David C.
Walford, Robert B.
Walker, Rob
Walker, Donald E.
Walston, Claude E.
Walters, T. L. Jr.
Wand, Mitchell
Ward, Jas. A.
Ward, Wayne D.
Warner, Walter P.
Wasserman, Anthony I.

Watson, W. J.
Wear, Larry L.
Weber, R. A.
Wedberg, George H.
Weihrer, Anna Lea
Weingarten, Fred W.
Weirte, William M.
Weiss, Edward C.
Weiss, Eric A.
Weiss, Stephen F.
Welker, Nancy K.
Wells, James M.
Wells, Mark B.
Wen, K. Y.
Wensley, John H.
Wesselkamper, T. C.
Westin, Alan F.
Wexelblat, R. L.
Weyl, Stephen
Wheeler, T. F.
Whinston, Andrew
White, George M.
White, John R.
Whitman, Kirwin A.
Whitney, Daniel E.
Wieselman, Irving L.
Wile, David S.
Willett, Richard M.
Williams, Leland H.
Williams, Charles M.
Williams, Richard P.
Williams, Theodore J.
Williams, Thomas G.
Willman, Herbert E. Jr.
Wilner, Wayne T.
Wilson, Edwin B.
Wiorkowski, Gabrielle
Wofsey, Marvin M.
Wolf, Eric W.
Wood, David B.
Woodbury, Max A.
Woodson, M. I. Chas. E.
Wooster, Harold
Wooton, Leland M.
Worlton, Jack
Wright, Charles
Wright, Kendall R.
Wright, S. E.
Wulf, Wm. A.
Wyllys, Ronald E.
Wyner, Donald S.

Yamada, Gordon T.
Yao, S. B.
Yarbrough, Lynn
Yonda, A. W.
Young, J. W.
Yovits, Marshall C.

Zelkowitz, Marvin V.
Zellweger, Andres G.
Zilles, Stephen N.
Zimmerman, Joan
Zimmerman, Martin B.
Zinn, Karl L.
Zislis, Paul M.
Zosel, Mary E.
Zungoli, S. S.

AUTHOR INDEX